Introduktion till Matlab

Algebra och analys för högskoleingenjörer

Jens Fjelstad

24 oktober 2019

Innehåll

1 Introduktion

Syftet med den här texten är att ge en introduktion till att använda programvaran Matlab för att lösa vissa typer av matematiska problem i kursen *Algebra och analys för högskoleingenjörer* på Örebro universitet. Det förutsätts inte att du har någon som helst erfarenhet av programmering eller av att använda datorn för matematik, men hur fort du kan och vill gå framåt kan förstås bero på tidigare erfarenheter. Allt du behöver för att arbeta dig igenom materialet finns i en kombination av texten och Matlabs dokumentation, och du ska därför kunna arbeta med övningarna även på egen hand utanför övningstid.

Matlab finns installerat i teknikhusets datorsalar. Som student har du dessutom möjlighet att kostnadsfritt installera Matlab på din egen dator; se information här. Jag råder dig starkt att köra Matlab på din egen dator om du alls har möjlighet att göra så.

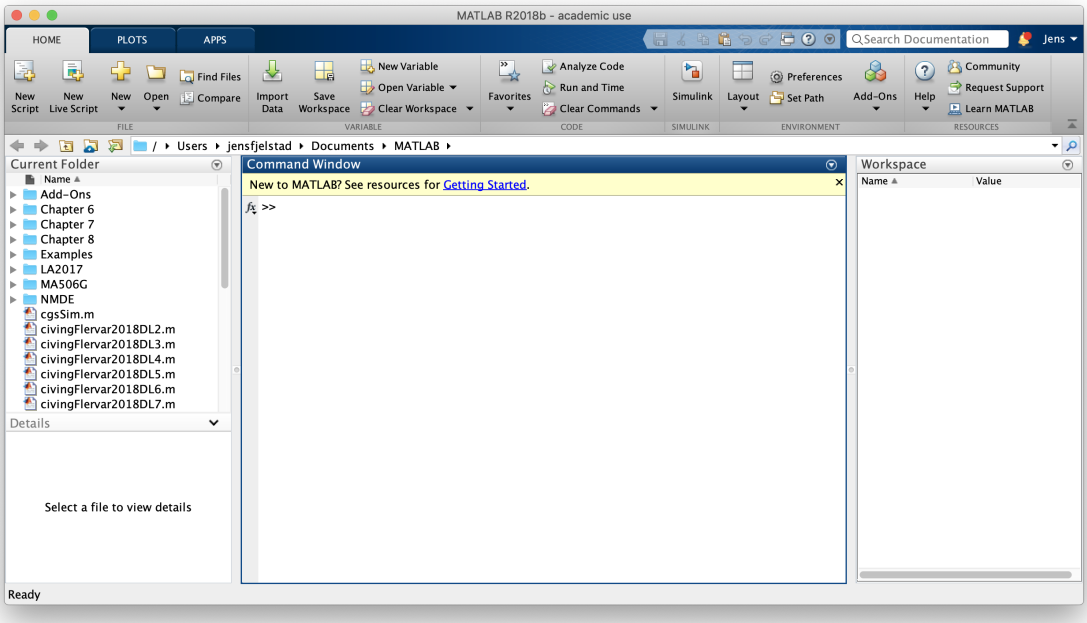
Slutligen några ord om hur du bäst använder denna text. Lämpligtvis sitter du och läser framför en dator som kör Matlab, och när du kommer till exempel på hur ett kommando eller liknande används i Matlab så försöker du att följa exemplet i texten i ditt Matlabfönster. När du arbetat dig igenom texten försöker du dig på de följande, obligatoriska, övningarna.

2 Matlab som miniräknare

I detta första avsnitt ska vi diskutera grundläggande saker som de olika delarna av Matlabfönstret, hur du kommunicerar med Matlab i den s.k. Kommandotolken, och vad som menas med variabler.

2.1 Matlab när det körs för första gången

Starta Matlab. Matlab öppnar ett fönster som ser ut något i stil med bilden i Figur 1. Ditt fönster kommer inte att vara identiskt med mitt, det kan skilja sig på flera olika sätt men säkert är att den aktuella mappen (Current Folder) inte kommer att heta samma sak eller innehålla samma filer. Om det är första gången du startar Matlab kommer



Figur 1: Ett Matlabfönster med kommandotolk (Command Window) i mitten, ett fönster med innehållet i den aktuella mappen (Current Folder) till vänster, ett fönster kallat Workspace till höger, ett verktygsfält med klickbara ikoner, samt en samling flikar längst upp.

mappen antagligen att vara helt tom. Längst upp finns en stor mängd knappar, dessa (eller åtminstone några av dem) får vi tillfälle att återkomma till senare. Dessutom finns (åtminstone) tre separata fönster med namn som Current Folder, Command Window eller *kommandotolk*, och Workspace. Detta avsnitt handlar om de två sista av dessa.

2.2 Att använda Matlab som miniräknare

Matlab kan t.ex. användas som en miniräknare. Placera markören i kommandotolken, skriv `1+1` och tryck på `Enter`. Om du har gjort rätt svarar Matlab med `ans = 2`. Den här proceduren, att skriva något i kommandofönstret och sedan trycka på `Enter` kommer jag i fortsättningen kalla *att ge ett kommando*. Hela konversationen i kommandotolken ser ut som följer.

```
>> 1+1

ans =

     2
```

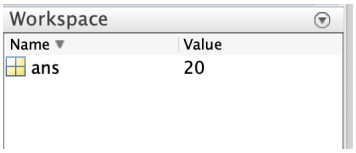
Uttrycket `ans` är förstås tänkt att påminna om ordet *answer*, så Matlab svarar således att svaret är lika med 2. Faktum är att den precisa betydelsen är något annorlunda. Ge nu kommandot `ans` (dvs skriv `ans` och tryck på `Enter`). Resultatet är

```
>> ans

ans =

     2
```

Hur ska detta tolkas? Matlab har skapat en s.k. *variabel* med namnet `ans` och värdet 2. Varje operation som Matlab utför uppdaterar variabeln `ans` med värdet på operationen. Kommandot `4*5` resulterar i

```
>> 4*5

ans =

     20
```

Betrakta fönstret *Workspace*, det visas i Figur 2. I detta fönster listas namn och värde



Figur 2: Workspace-fönstret efter den senaste beräkningen. En variabel med namn `ans` finns lagrad, dess värde är 20.

på de variabler som har definierats.

Språkbruket stämmer väl överens med hur vi använder symboler x, y, etc. som namn på variabler i matematik. Variabler representerar, ofta godtyckliga, tal som vi ibland vill välja att anta specifika värden.

Vi kan definiera andra variabler, t.ex. resulterar kommandot `x=2/3` i en variabel med namnet `x` och värdet 0.6667.

```
>> x=2/3

x =

    0.6667
```

När vi skriver `x=2/3` tolkar Matlab det som att vi definierar en variabel med namnet `x`, och tilldelar den värdet 0.6667. Vi kan nu använda x i beräkningar som följande exempel visar.

```
>> x*x/2

ans =

     0.222
```

Kommandot `who` ger namnen på de variabler som är definierade, och kommandot `whos` ger dessutom annan information om alla variabler.

```
>> who

Your variables are:

ans x

>> whos
    Name        Size              Bytes  Class     Attributes

    ans         1x1                   8  double
    x           1x1                   8  double
```

Vad denna information betyder lämnar vi till ett senare tillfälle.

Hur ska vi förstå decimaltalet 0.6667 ovan? Det är förstås en approximation till talet 2/3, men varför visar Matlab just denna approximation och inte en med, säg, 2 eller 10 decimaler? Matlab själv lagrar betydligt fler decimaler. Kommandot `format long` tvingar Matlab att skriva ut fler decimaler

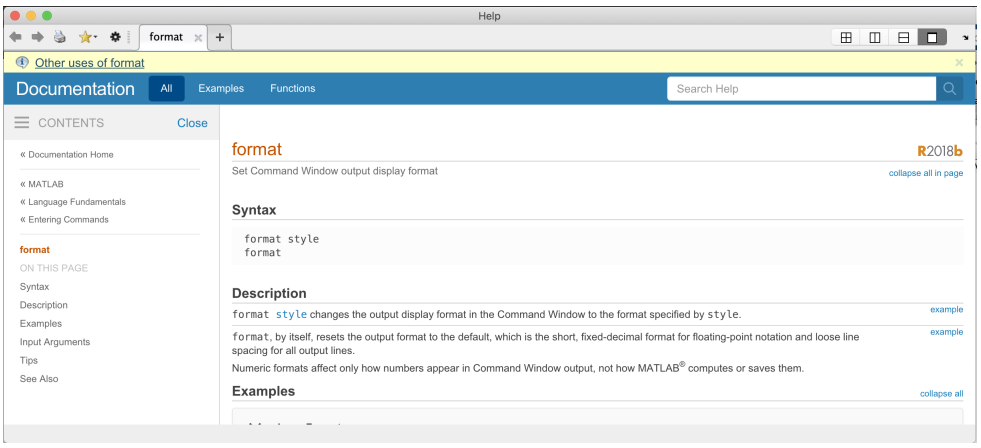
```
>> format long
>> x

ans =

     0.666666666666667
```

Gå tillbaka till det ursprungliga formatet via kommandot `format`. Det finns också en mängd andra format i Matlab.

Ett viktigt verktyg i att lära sig använda Matlab är Matlabs egen hjälpfunktion och dokumentation; där finns väldigt mycket användbar information och exempel. För att se vad Matlabs hjälpfunktion säger om olika format, ge kommandot `help format` (pröva). Du får information om varje kommando på samma sätt (`help` följt av kommandots namn). Än mer användbar är Matlabs dokumentation vilken du kommer åt genom att ersätta `help` med `doc`. Kommandot

```
>> doc format
```

öppnar en browsersida med dokumentationen för begreppet `format`, se Figur 3. Tag för vana att först läsa dokumentationen om du undrar hur ett kommando fungerar, där finns dessutom ofta exempel på hur kommandot används.



Figur 3: Kommandot `doc format` öppnar detta browserfönster, som visar den information om begreppet `format` som finns i Matlabs dokumentation.

Som du kommer att upptäcka så kommer du att behöva spara resultatet av i princip varje beräkning i en variabel (t.ex. för att kunna använda resultatet i fortsatta beräkningar). Om du behöver göra många beräkningar kan det därför vara praktiskt att kunna tilldela variabler beräkningarnas värden utan att varje gång skriva ut resultatet. Det åstadkommer du genom att avsluta kommandot med semikolon, t.ex. som i

```
>> y=2+3/2;
```

En blick på Workspacefönstret säger oss att en ny variabel `y` har definierats. Vi kan se dess värde genom att ge kommandot `y`.

```
>> y

y  =

    3.5000
```

Uppenbarligen har Matlab tolkat kommandot `2+3/2` som *addera* 2 *till resultatet av att dividera* 3 *med* 2, vilket ju är precis samma tolkning som vi gör. Det gäller också mer allmänt att Matlab har samma prioriteringsregler för beräkningar med addition, subtraktion, multiplikation och division, som vi själva använder oss av. Om vi istället först vill addera 2 med 3, för att sedan dividera resultatet med två gör vi det precis som för hand, genom att sätta ut parenteser på lämpliga ställen. Se t.ex.

```
>>z=(2+3)/2

z =
    2.5000
```

Matlab kan mer än de fyra räknsätten. Potenser beräknas med symbolen ^, t.ex. beräknas 2^3 via kommandot `2^3`.

```
>> 2^3

ans =

     8
```

Matlab har också ett stort antal inbyggda funktioner, t.ex. trigonometriska funktioner och logaritmer men också många andra. Den naturliga logaritmen heter `log` och sinusfunktionen heter `sin`. Se följande exempel.

```
>> log(1)

ans =
    0

>> exp(1)

ans =

    2.7183

>> exp(log(3))

ans =

    3

>>sin(pi/2)

ans =

    1
```

I det sista exemplet används det fördefinierade talet π, vilket anges som `pi`. Ge kommandot för att få π med fyra decimaler. Kan du utifrån exemplet gissa vad kommandot `exp(x)` gör (x representerar ett godtyckligt tal)?

Variabler i Matlab har både likheter och skillnader med de variabler vi använder i matematik när vi arbetar med papper och penna. En skillnad som jag vill poängtera har att göra med hur Matlab tolkar en variabel, säg `x`, som redan är tilldelad ett värde. Studera följande exempel.

```
>> x=7;
>> y=2*x

y =
```

```
    14

>> x=3;
>> y

y =
    14
```

Den här följden av kommandon definierar först variabeln `x` och tilldelar den värdet 7, sedan definieras variabeln `y` som tilldelas värdet $2x = 14$, efter detta definieras variabeln `x` om och tilldelas nu värdet 3, och till sist kontrollerar vi igen vad `y` har för värde, med samma resultat som tidigare, 14. Observera att variabeln `y` definierades att ta värdet `2*x`, men värdet på `y` *ändras inte* trots att värdet på `x` ändras. Anledningen är att när vi ger kommandot `y=2*x` så ersätter Matlab omedelbart `x` med dess värde (i det här fallet 7); det ska alltså inte tolkas som att `y` alltid tar värdet `2*x` oavsett vad `x` tar för värde[1], utan snarare som att `y` tar värdet `2*` det värdet som variabeln `x` har i det ögonblicket vi definierar `y`.

2.3 Att göra fel

Råkade du skriva fel vid något tillfälle när du följde kommandona i det örra avsnittet? Du är i så fall i gott sällskap, det händer de flesta. Eventuellt har du då också upptäckt att Matlab är väldigt känsligt för precis hur du skriver kommandon osv. Detta gäller programmeringsspråk i allmänhet, inte bara Matlab. För att Matlab ska förstå vad du skriver krävs att du skriver exakt rätt. T.ex. skiljer Matlab på gemener och versaler, så om du råkar skriva `Sin(pi/2)` istället för `sin(pi/2)` så får du ett felmeddelande. I detta, och många andra, fall så gissar Matlab vad du egentligen ville skriva och frågar detta.

```
>> Sin(pi/2)
Undefined function or variable 'Sin'.

Did you mean:
>> sin(pi/2)
```

Något liknande kan hända om du t.ex. glömmer bort att skriva tecknet `*` för att indikera multiplikation. Testa att skriva `xx` och `sin(x)sin(x)` istället för `x*x` respektive `sin(x)*sin(x)`.

Det finns otaliga sätt att skriva fel på, och syftet med det här avsnittet är inte att försöka ta upp dessa, eller ens de vanligaste. Syftet är istället att förmedla att när Matlab ger ett felmeddelande eller inte fungerar som du har tänkt så beror det med största sannolikhet på att du har skrivit fel någonstans. Det första du bör göra är alltså att gå igenom vad du skrev i detalj och jämföra med hur det ska vara. Ibland kan detta vara oerhört frustrerande då det kan vara svårt att hitta fel, men det är också något

[1] Det är å andra sidan möjligt att använda variabler mer likt hur vi är vana att använda dem, symboliskt. Detta återkommer vi till i avsnitt 5.

som går att bli bättre på med lite vana (både att undvika att skriva fel, och att hitta uppkomna fel).

2.4 Sammanfattning

▶ Matlab betecknar de fyra räknesätten $+$, $-$, $*$, och $/$. En potens anges med `^`.
Ex: `5+3`, `5-3`, `5*3`, `5/3`, `5^3`.

▶ `x=2` definierar en *variabel* `x` och ger denna *värdet* 2.

▶ Kommandona `who` och `whos` ger information om alla variabler som definierats. Information om dessa återfinns också i Workspace-fönstret.

▶ För att läsa Matlabs dokumentation ge kommandot `doc` följt av funktionen eller begreppet du vill läsa om. Matlabs hjälpfunktion i Kommandofönstret anropas på samma sätt men med `help` istället för `doc`.

▶ `format long` medför att tal skrivs ut med 16 värdesiffror (ibland faktiskt 8, läs om *single values* och *double values* i dokumentationen) i kommandofönstret. `format` får Matlab att återgå till det korta formatet. Läs dokumentationen för att se vilka andra format som kan anges.

▶ Semikolon `;` efter ett kommando medför att resultatet av kommandot inte skrivs ut.

▶ Matlabs prioritetsordning mellan olika operationer följer samma konventioner som du har lärt dig i skolan (multiplikation "binder starkare" än addition t.ex.). Ibland krävs därför att vi sätter ut parenteser för att kräva att Matlab beräknar saker i rätt ordning.

▶ Matlab har vissa fördefinierade tal, t.ex. talet $\pi = 3.1416$.

2.5 Övningar

2.1 Ge ett Matlabkommando som beräknar följande tal, i alla de beskrivna stegen. I båda deluppgifterna, skriv först ner uttrycket som det beskrivs i ord med papper och penna.

(a) kvoten av talet a med talet b, där a ges av att först subtrahera 51 från 276 och sedan multiplicera resultatet med 33, och b är produkten av 3^3 med 55.

(b) samma kvot, men där talet a istället ges av att subtrahera 51 från produkten av 276 och 33.

2.2 Definiera en variabel med namnet `theta` och värdet $\pi/3$, och beräkna cosinus av `theta`. Observera att vi inte har tagit upp cosinus-funktionen i den föregående texten. Kanske kan du ändå gissa hur den anges? Om inte, kan du komma på något sätt att försöka ta reda på det?

2.3 Definiera variabeln `e` att ta värdet hos talet e med de fyra första decimalerna. Beräkna den naturliga logaritmen av din variabel `e` och bestäm med fyra decimalers noggrannhet skillnaden mellan detta och 1.

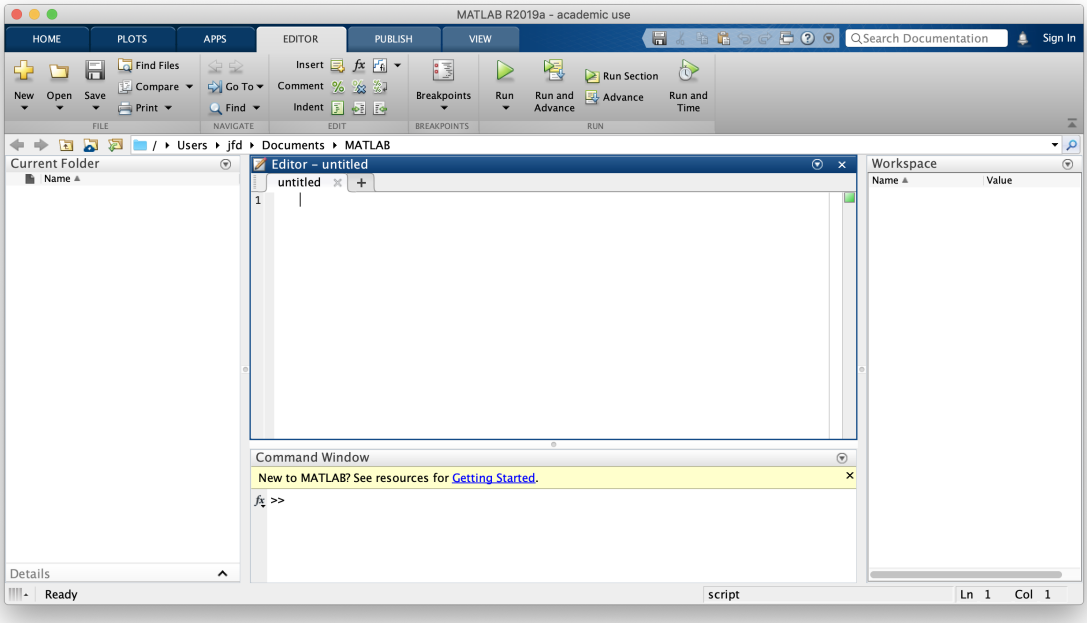
3 Funktionsgrafer och script

Vi fortsätter med att lära oss hur Matlab kan användas för att illustrera funktioner via deras grafer, men först tittar vi snabbt på *script* vilket är vad som används för att programmera i Matlab, dvs att utföra successiva kommandon i ordningsföljd.

3.1 Matlabs script

Så gott som alltid vill vi låta Matlab utföra mer än en sak, typiskt flera kommandon där resultatet av ett ska ingå som input till ett annat. Dessa situationer blir svårhanterliga om vi fortsätter att arbeta i Kommandotolken som i det första avsnittet. Det vanliga är istället att skriva program som sparas i filer. I Matlabs terminologi kallas ett sådant program *script*, och filerna kallas ofta *m-filer* (eftersom filtillägget är `.m`).

Ett script skrivs t.ex. i Matlabs egen texteditor, som öppnas genom kommandot `edit` eller genom att klicka på knappen *New Script* under fliken *Home*. På min dator resulterar det i följande utseende. Notera den tomma texteditorn i mitten.



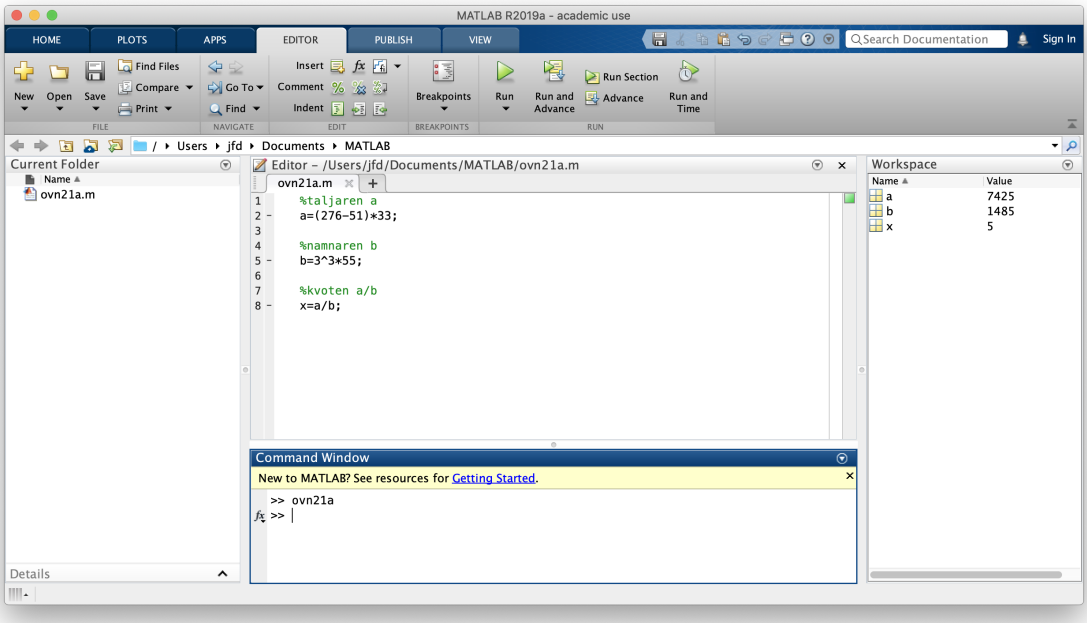
Figur 4: Matlab med tomt Editorfönster i mitten.

Ett script består helt enkelt av en följd av Matlabkommandon, ordnade uppifrån och ner. När scriptet "körs" så utförs kommandona i ordningsföljd. Säg att vi vill lösa övning 2.1. (a) genom att först definiera variablerna a och b, och sedan spara resultatet i en variabel x. Ett script som utför detta är följande.

```
%taljaren a
a=(276-51)*33;

%namnaren b
b=3^3 * 55;

%kvoten a/b
x=a/b;
```

Notera texten i grönt som börjar med tecknet `%`; all text som följer fram till nästa radbyte (dvs all text som visas i grönt i editorn) ignoreras av Matlab. Detta används för att infoga *kommentarer* i script. Tag för vana att *alltid* lägga in kommentarer för att förklara vad olika delar av koden gör i ett script, formulera dig så att det går att förstå scriptet om du återvänder till det ett år senare.

För att köra scriptet måste det först sparas i en fil; gör detta genom att klicka på knappen *Save* under fliken *Editor*. Du får välja ett filnamn, jag valde namnet "ovn21a" vilket medför att det i Current Folder-mappen dyker upp namnet `ovn21a.m`. Notera att filen *måste* sparas i någon av mapparna som Matlab letar i för att kuna köras. Den mappen som är "Current Folder" när du först startar Matlab, och varje mapp som du skapar i den mappen, fungerar bra att spara filer i. Scriptet körs nu antingen genom att klicka på knappen *Run*, eller genom att i Kommandotolken ge kommandot `ovn21a`. Efter att ha kört scriptet ser Matlabfönstret ut så här. Vi ser tecken på att scriptet



Figur 5: Matlabfönstret efter att ha kört scriptet `ovn21a.m`.

har körts i Kommandotolken (kommandot `ovn21a`) och i Workspace-fönstret där vi

identifierar variablerna `a`, `b`, och `x` med sina värden. Resultatet är identiskt med om vi hade gett var och ett av kommandona för hand i Kommandotolken, och resultatet kan förstås användas vidare på samma sätt.

Ibland är det användbart att kunna köra endast en del av ett script. Kanske vi i exemplet ovan vill kunna byta `x=a/b` till `x=b/a`, men för att göra detta är det inte nödvändigt att beräkna `a` och `b` på nytt. Placerar vi ett dubbelt %-tecken i koden så förstår Matlab det som att tecknet separerar koden i två delar, där var och en av delarna kan köras separat. I det tänkta exemplet har vi följande script.

```
%taljaren a
a=(276-51)*33;

%namnaren b
b=3^3 * 55;

%%
%kvoten b/a
x=b/a;
```

Placera markören någonstans från separationstecknet och ned, den nedre delen av scriptet markeras då med gult. Kör den markerade delen genom att klicka på *Run Section*. Du kan placera ut hur många separationstecken som helst i ett script, och varje del mellan två närliggande separationstecken kan köras separat.

3.2 Rita grafen till $f(x)$

Under kursmomentet *Differential- och integralkalkyl* kommer du att lära dig mer om att skissa grafen till en funktion. Redan nu kan vi å andra sidan vilja se hur grafen till en funktion ser ut, inte minst när det gäller funktioner av flera variabler. En av de första uppgifterna blir därför att använda Matlab till att rita funktioners grafer.

Låt $f(x)$ vara en funktion. Vi lär oss tidigt att skissa grafen till f på ett intervall $[a, b]$ som tillhör f:s definitionsmängd genom att välja några tal x_1, x_2, ..., x_n i intervallet $[a, b]$, och göra en tabell över funktionsvärdena i dessa tal.

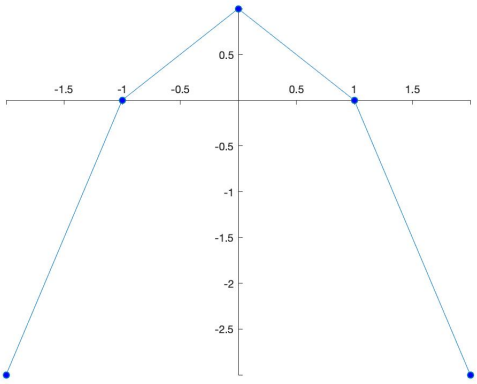
x	x_1	x_2	$\cdots$	x_n
$f(x)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

Vi markerar sedan punkterna $(x_i, f(x_i))$ i ett koordinatsystem, förbinder punkterna med linjer och hoppas att resultatet är en bra approximation till f:s graf.

Vi tar det konkreta exemplet att skissa grafen till $f(x) = 1 - x^2$ för $x \in [-2, 2]$. Välj punkterna -2, -1, 0, 1, 2 och beräkna funktionsvärdet i var och en av dessa, dvs $f(-2) = 1 - (-2)^2 = -8$, $f(-1) = 1 - (-1)^2 = 0$, $f(0) = 1 - 0^2 = 1$, $f(1) = 1 - 1^2 = 0$, $f(2) = 1 - 2^2 = -3$. För fullständighetens skull tabellerar vi också resultatet.

x	-2	-1	0	1	2
$f(x)$	-3	0	1	0	-3

Grafen vi skissar kommer att se ut något i stil med



Eventuellt vågar vi oss på att göra grafen något mjukare. De markerade punkterna är de som anges i tabellen, dvs (från vänster till höger) $(-2,-3)$, $(-1,0)$, $(0,1)$, $(1,0)$, och $(2,-3)$.

Matlab kan rita grafer till funktioner på samma sätt, dvs genom att beräkna funktionsvärden i ett ändligt antal punkter, rita ut punkterna i planet och förbinda dessa med linjer. Säg att vi, som i exemplet ovan, vill rita grafen till funktionen $f(x) = 1 - x^2$, $x \in [-2, 2]$. Välj först ut ett antal värden på x, t.ex. alla punkter mellan -2 till 2 med steg om 0.1: -2, -1.9, -1.8, ..., 1.9, 2.

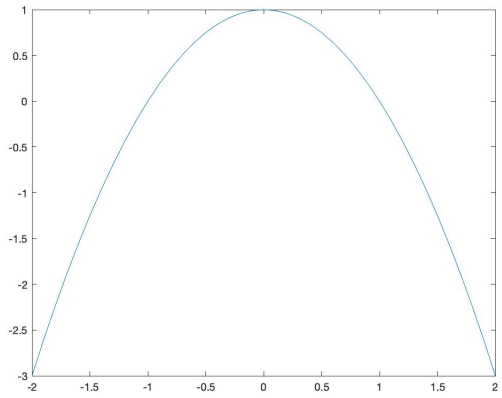
Följande script ritar grafen till f genom att sammanbinda punkterna $(x, f(x))$ där x tar dessa värden.

```
x=-2:0.1:2;      %lista med varden pa variabeln x
y=1-x.^2;        %lista med funktionsvarden f(x)
plot(x,y)        %forbinder punkterna med koordinater ur x och y med linjer
```

Vi sparar scriptet under namnet `graf.m` och kör det, resultatet kan ses i Figur 6. Det finns många saker att diskutera kring exemplet. Låt oss först gå igenom scriptet rad för rad.

Den första raden definierar en variabel x som är en lista med alla tal mellan -2 och 2 med steg om 0.1. Ett annat ord för en sådan lista, och ordet vi ska använda, är *vektor*. Det finns många sätt att konstruera vektorer i Matlab, ett annat sätt att definiera precis samma vektor är via kommandot `x= linspace(-2,2,41)`. Här anges istället för steglängden, antalet element i vektorn.

Den andra raden definierar en vektor y med alla funktionsvärden. Uttrycket i högerledet härrör förstås från $1 - x^2$ i funktionens definition, men notera punkten framför symbolen `^2` som indikerar kvadraten av x. Den punkten talar om för Matlab att utföra operationen som följer (dvs att kvadrera) på *varje element* av vektorn x, och att stoppa in resultaten i en ny vektor. För att sammanfatta, resultatet av den andra raden är en ny vektor y där första elementet är resultatet av att beräkna 1 minus kvadraten av första



Figur 6: Grafen till funktionen $f(x) = 1 - x^2$ ritad av Matlab i 41 punkter mellan -2 och 2.

elementet av vektorn x, andra elementet ges av 1 minus kvadraten av andra elementet av vektorn x, etc.

Den tredje raden slutligen talar on för Matlab att rita ut punkterna i ett koordinatsystem i planet vars x-koordinater är elementen i vektorn x, och vars y-koordinater är elementen i vektorn y, i ordning från det första till det sista elementet, samt att sammanbinda närliggande punkter med linjer. Eftersom vektorn y innehåller funktionsvärdena så är punkternas koordinater precis $(-2, f(-2))$, $(-1.9, f(-1.9))$, ..., $(2, f(2))$.

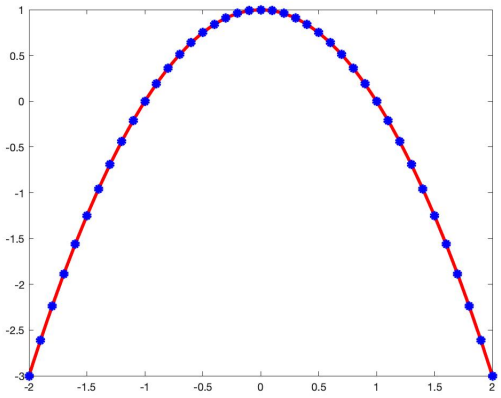
Figuren, som öppnats i ett eget fönster, visar x-axeln längst ner och y-axeln längst till vänster, och kommer alltid att göra så om vi inte instruerar Matlab att göra något annat. Med andra ord, koordinataxlarna i Matlabs figurer skär varandra vanligtvis inte i origo. Det finns många sätt vi kan ändra utseendet på en figur i Matlab. Kommandot

```
plot(x,y,'-o','Color','r','LineWidth',3,'MarkerEdgeColor','b','MarkerFaceColor','b')
```

ger t.ex. resultatet som visas i Figur 7. Kan du gissa vad alla tillägg i plot-kommandot gör? En liten ledtråd är att antalet "prickar" i grafen är precis 41. Läs dokumentationen för `plot` om du vill veta mer, och dessutom lära dig om många andra sätt att ändra utseendet på en figur. Tillägg i kommandon kallas ofta *växlar* på svenska, *options* på engelska.

3.3 Flera grafer samtidigt?

När du har jobbat med grafer i det förra avsnittet så har du säkert upptäckt att varje gång du k¨or ett `plot`-kommando så rensas figurfönstret och endast den senaste grafen visas. Ibland kan det dock vara användbart att visa flera grafer i ett och samma figurfönster, Ett enkelt sätt att göra detta är m h a kommandot `hold on` (läs dokumentationen för

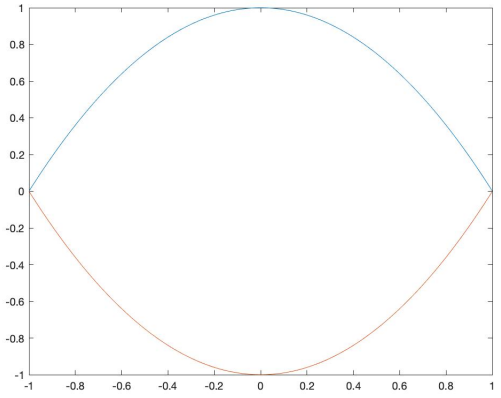


Figur 7: Samma graf som i Figur 6, men formatterad annorlunda.

`hold`). Säg att vi i samma fönster vill visa graferna för $1 - x^2$ och $x^2 - 1$ på samma intervall $[-1, 1]$. Följande kod gör utför detta

```
x=-1:0.01:1;     %lista med varden pa variabeln x
y=1-x.^2;    %varden hos funktionen 1-x^2 i punkterna x
z=x.^2-1;    %varden hos funktionen x^2-1 i punkterna x

plot(x,y)    %rita grafen till funktionen 1-x^2 mellan x=-1 och x=1

hold on %behall innehallet i figurfonstret varje gang en ny graf ritas

plot(x,z)    %rita grafen till funktionen x^2-1 mellan x=-1 och x=1
```

Resultatet syns i Figur 8.



Figur 8: Graferna till $1 - x^2$ och $x^2 - 1$ i en och samma figur.

Vad händer nu om vi ritar fler grafer? Testa!

Varje ny graf kommer att ritas i samma figur. Så sker tills Matlab får kommandot `hold off`, eller figurfönstret stängs.

3.4 Mer om vektorer

Vi har sett två sätt att definiera vektorer, här kommer ett till skrivet i kommandotolken.

```
>> x=[1, 2.3 , -5]

x =

    1.0000    2.3000   -5.0000
```

Kommandot definierar en vektor med första element 1, andra element 2.3, och tredje element -5. Vi noterar att alla talen anges på decimalform, men bryr oss f.n. inte om detta. Mellanrummen efter kommatecknen är inte nödvändiga, och i själva verket är inte heller kommatecknen nödvändiga (men i avsaknad av dessa krävs förstås mellanrum mellan elementen). Om vi vill använda oss av det första elementet i vektorn x kan vi ange det genom kommandot `x(1)`. På samma vis anger vi det andra elementet som `x(2)`, tredje som `x(3)` etc.

Definiera nu vektorn `y=[-1, -1, 3]`. I det förra avsnittet såg vi att vi enkelt kan kvadrera en vektor x elementvis, dvs beräkna en ny vektor där elementen är kvadraterna av elementen i vektorn x. Detta generaliserar till många andra operationer. T.ex. kan vi addera och multiplicera två vektorer x och y, multiplicera vektorer med tal, och beräkna funktionsvärden av fördefinierade funktioner elementvis. Vi testar i kommandofönstret.

```
>> x+y

ans =

         0    1.3000   -2.0000

>> x.*y

ans =

   -1.0000   -2.3000  -15.0000

>> 3*y

ans =

    -3    -3     9

>> x.^y
```

```
ans =

    1.0000    0.4348 -125.0000

>> sin(x)

ans =

    0.8415    0.7457    0.9589

>> x./y


ans =

   -1.0000   -2.3000   -1.6667
```

Notera att det endast är för operationerna `*`, `/`, och `^` som vi behöver skriva en punkt framför operationen. Du kommer senare i kursen att förstå varför dessa operationer måste behandlas speciellt. Studera resultaten och övertyga dig själv om att var och en av operationerna har skett elementvis, dvs på var och ett av vektorernas element för sig, vilket ger en ny vektor med lika många element som den eller de som användes i operationen.

Det bör kännas naturligt att i varje operation ovan som inbegriper två vektorer så måste de ha lika många element. Testa att addera två vektorer med olika antal element. Vad händer? I felmeddelandet du får dyker ordet *matrix* upp (*matris* på svenska). Vektorer är exempel på en typ av objekt som kallas matriser och som vi ska studera i detalj under kursens gång. Redan i nästa avsnitt så måste vi använda oss av mer allmäna matriser dock.

3.5 Sammanfattning

- Texteditorn öppnas med kommandot `edit` eller knappen *New script*
- Kommentarer infogas i skript med tecknet `%`
- Tecknet `%%` delar in ett script i delar som kan köras separat
- `x=linspace(a,b,N)` definierar en vektor x med N element där det första är a, det sista är b, och steget mellan närstående element är lika stort överallt. Samma vektor erhålles via `x=a:(b-a)/(N-1):b`.
- Operationerna `*`, `/`, och `^` utförs elementvis på vektorer om en punkt placeras innan operationen, t.ex. $x.*y$.
- `plot(x,y)` ritar grafen av funktionen $f(x)$ sådan att $y(n) = f(x(n))$ för varje element $x(n)$ i x.
- `hold on` säger åt Matlab att lägga till kommande grafer i den befintliga figuren istället för att först radera den. Stäng av funktionen genom kommandot `hold off`.
- De flesta fördefinierade funktionerna verkar elementvis på vektorer.

3.6 Övningar

Från och med dessa övningar så ska du spara alla lösningar som script i en fil.

3.1 Använd Matlab till att rita grafen till funktionen $f(x) = x^3 - 5x^2 + 3$ på ett intervall som du väljer själv. Försök att hitta ett intervall där funktionen f har tre skilda nollställen.

3.2 Hur många lösningar har följande ekvationer? Bestäm grafiskt.

(a) $\sin(x) = \cos(x)$ där $x \in [\pi/3, 7\pi/6]$

(b) $\tan^2(\alpha) = 3\sin(3\alpha)$ där $-\dfrac{\pi}{3} \leq \alpha \leq \dfrac{\pi}{3}$

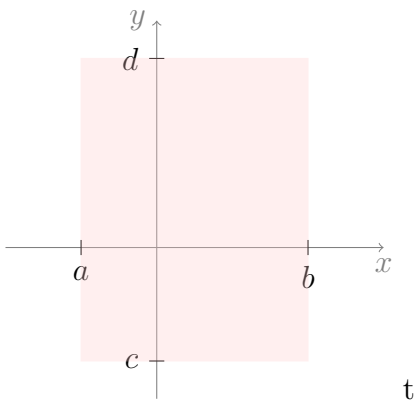
3.3 Rita grafen till funktionen $g(\theta) = \sqrt{2}\sin(\theta) - \sqrt{2}\cos(\theta)$. Verkar det rimligt att funktionen kan skrivas som $A\sin(\theta + \phi)$ för något värde på A och på ϕ? Försök finna exakta sådana värden genom att jämföra grafer.

4 Funktioner av flera variabler

Som vi har berört ytligt i teoridelen av kursen så finns flera sätt att illustrera funktioner av flera variabler. Vi begränsar oss till funktioner av två variabler, och går igenom hur vi ritar graf och nivåkurvor till en sådan funktion.

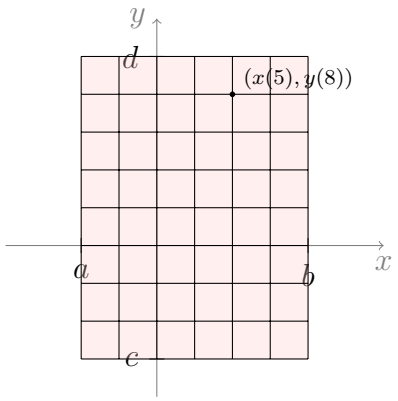
4.1 Rita grafen till $f(x, y)$

Vi har inte ägnat mycket energi i teoridelen åt att rita grafer till funktioner av två variabler, det återkommer dock senare i den andra delen av kursen. Det finns å andra sidan ingen orsak att inte använda Matlab för att rita sådana grafer. Vi antar att $f(x, y)$ är en funktion vars graf vi vill rita på rektangeln $a \leq x \leq b$, $c \leq y \leq d$. För att förtydliga, vi är alltså intresserad av grafen till funktionen f på området som består av alla punkter (x, y) som täcker en rektangel med sidor parallella med koordinataxlarna. Följande figur får illustrera denna rektangel.



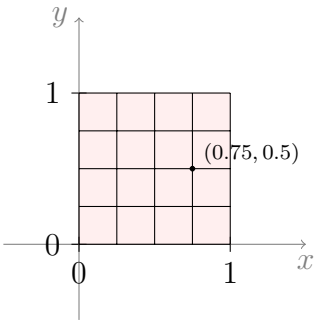
För en funktion $f(x)$ av en variabel utgick vi från ett ändligt antal punkter. Vi gör samma sak här. För att skissa grafen till $f(x, y)$ så väljer vi ut ett ändligt antal punkter i rektangeln, beräknar funktionsvärdet i var och en av dessa, ritar ut punkterna i tre dimensioner ovanför rektangeln där höjden anges av funktionsvärdet, och slutligen ansluter vi punkterna med någon form av yta. Metoden är inte lätt att använda för hand, men för Matlab är det inga problem.

För att välja ut ett antal punkter i rektangeln så väljer vi först ett visst antal punkter i intervallet $[a, b]$ längs x-axeln och ett visst antal punkter i intervallet $[c, d]$ längs y-axeln. Vi tänker oss sedan ett rutnät med punkter vars koordinater är just dessa möjliga värden på x respektive y.



I figuren ovan har vi valt 7 värden i $[a, b]$ på x-axeln och 9 värden i $[c, d]$ på y-axeln. En punkt i rutnätet är markerad med en fylld cirkel och koordinater.

Matlab använder sig av ett rektangulärt schema med tal för att lagra tal som t.ex. koordinater och funktionsvärden i varje punkt på rutnätet. Ett sådant rektangulärt schema av tal kallas *matris*. Låt oss ta ett konkret exempel, kvadraten $0 \leq x \leq 1$, $0 \leq y \leq 1$, där vi väljer fem punkter i varje intervall nämligen 0, 0.25, 0.5, 0.75 och 1. Definiera vektorerna `x=[0, 0.25, 0.5, 0.75, 1]`, `y=x`.



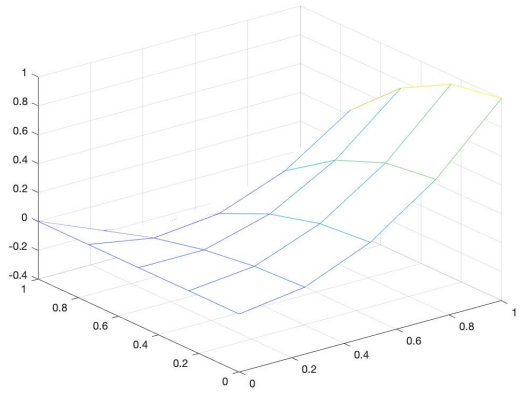
Vi definierar nu matriser X och Y, var och en med 5 rader och 5 kolumner, på följande sätt. Tänk på positionerna i en matris som en bild av punkterna på rutnätet. X innehåller då x-koordinaterna, och Y innehåller y-koordinater, för motsvarande punkter

på rutnätet, Kommandot `[X,Y]=meshgrid(x,y)` gör just detta. Matriserna X och Y ser ut som följer

$$X = \begin{pmatrix} 0 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \\ 0 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \\ 0 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \\ 0 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \\ 0 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 \\ 0.5000 & 0.5000 & 0.5000 & 0.5000 & 0.5000 \\ 0.7500 & 0.7500 & 0.7500 & 0.7500 & 0.7500 \\ 1.000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 \end{pmatrix}.$$

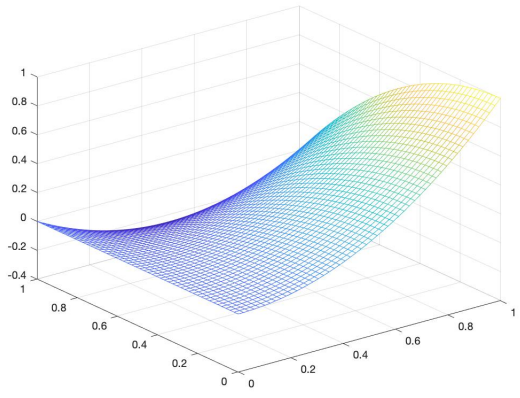
Elementet i rad nr 4 och kolumn nr 3 i matrisen X innehåller x-koordinaten 0.5 för punkten på rad nr 4 och kolumn nr 3 i rutnätet, osv. Matrisen Y inehåller p.s.s y-koordinaterna, men notera att den verkar upp-och-nervänd jämfört med bilden ovan. Värdet på y-koordinaten ökar när vi rör oss uppåt, medan den första raden i en matris är överst, och radnumret ökar nedåt. Därför blir matrisen felvänd jämfört med bilden av rutnätet.

Som ett exempel kan vi rita grafen till funktionen $f(x, y) = x^2 - xy^2$ på enhetskvadraten ovan. Vi behöver en till matris av samma storlek som X och Y, och vars element anger funktionsvärdet i motsvarande punkt på rutnätet. Vi definierar en sådan genom kommandot `Z = X.^2-X.*Y.^2`. Precis som för vektorer så anger vi elementvisa operationer på matriser med en punkt framför operationen (om den är `*`, `/`, eller `^`). Resultatet, Z, är en ny matris av samma storlek vars element i rad nummer r och kolumn nummer k ges av $x(k)^2 - x(k)y(r)^2$. Rita grafen via kommandot `mesh(X,Y,Z)`, vilket här ger Figur 9. Eftersom vi endast tagit fem punkter i varje koordinat blir grafen inte en särskilt



Figur 9: Grafen till $f(x, y) = x^2 - xy^2$ på kvadraten $0 \leq x \leq 1$, $0 \leq y \leq 1$, grovt rutnät

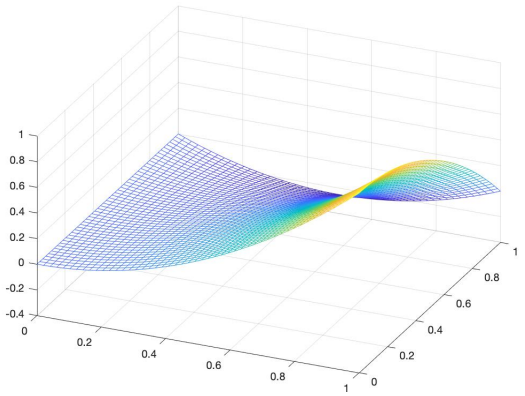
fin yta. Med 50 punkter i varje riktning får vi istället Figur 10.



Figur 10: Grafen till $f(x, y) = x^2 - xy^2$ på kvadraten $0 \leq x \leq 1$, $0 \leq y \leq 1$, fint rutnät

Försök återskapa Figur 10, hur går du tillväga för att använda 50 punkter i vardera riktning?

I en figur som avbildar grafen till en funktion av två variabler finns alltid ett perspektiv, vilket avgörs av betraktningvinkel och avstånd. Valet av perspektiv kan vara viktigt för en rättvisande bild av en graf. I Matlab kan perspektivet väljas på flera sätt, vi väljer att beskriva ett. Utgå från ett figurfönster med bild av en yta. Under "Tools"-menyn på figurfönstret, välj alternativet "Rotate3D". Om du placerar markören i figuren så ändras utseendet på markören. Håll nere den vänstra musknappen och drag, detta roterar figuren. Figur 11 visar samma graf som i Figur 10, men roterad för hand.

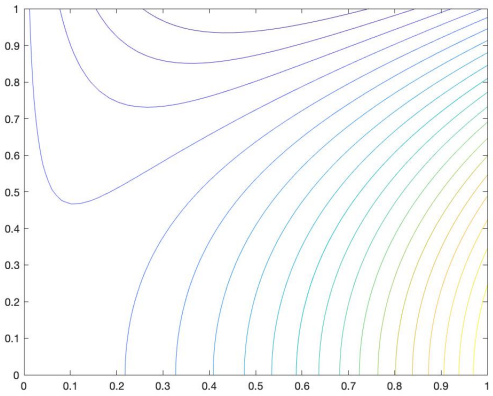


Figur 11: Samma graf som i Figur 10, men med annan betraktningsvinkel.

Ett annat kommando för att rita ytor är `surf`. Testa att byta ut `mesh` mot `surf` i något exempel ovan, kan du se någon skillnad?

4.2 Rita nivåkurvor till $f(x, y)$

För att ta fram nivåkurvor börjar vi på samma sätt som när vi ritar grafen till en funktion $f(x, y)$. Antag därför att vi har definierat matriser X Y, och Z som i förra avsnittet. Matlab ritar ett antal nivåkurvor genom kommandot `contour(X,Y,Z)`. Vill du ha precis 20 st nivåkurvor så skriver du istället `contour(X,Y,Z,20)`. Med funktionen $f(x, y) = x^2 - xy^2$ ger detta kommandot resultatet i Figur 12. I båda fallen väljer Matlab



Figur 12: Nivåkurvor, 20 st, för funktionen $f(x, y) = x^2 - xy^2$ i samma kvadrat som i Figur 10.

själv "nivåerna", dvs högerledet i $f(x, y) = k$.

Läs dokumentationen om `contour` och försök att få till följande figur. Du ska rita nivåkurvorna på nivåerna -0.2, 0, 0.1, 0.15, och 0.18, och samtidigt låta Matlab markera nivån för varje kurva i figuren.

4.3 Modifiera figurer

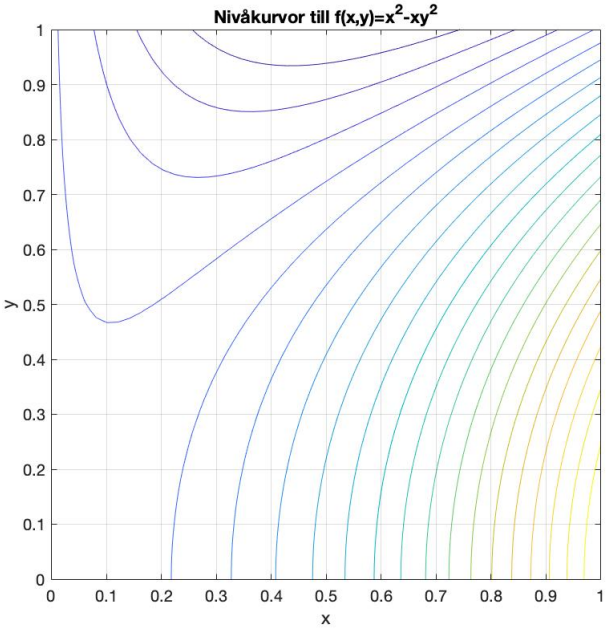
Precis som för kommandot `plot` så finns för `mesh`, `surf`, och `contour` ett antal växlar att lägga till för att ändra utseendet på grafen. Det finns också andra sätt att ändra figurer, både i två och tre dimensioner.

Koden som följer producerar bilden i Figur 13.

```
contour(X,Y,Z,20)
title('Nivakurvor till f(x,y)=x^2-xy^2')
xlabel('x')
ylabel('y')
axis equal
grid on
```

Antagligen är kommandona `title`, `xlabel`, `ylabel` och `grid` självförklarande. Läs om kommandot `axis` i dokumentationen. För olika sätt att justera ytor, kolla upp

`Surface Properties`. Det går förstås inte att ge en heltäckande bild av vad som kan



Figur 13: Samma nivåkurvor som i Figur 12, men med lite modifikationer.

göras och hur, istället vill jag försöka visa på sätt du kan lära dig själv på. Lär dig alltså att använda dokumentationen, det är väldigt användbart.

4.4 Sammanfattning

- En *matris* är ett rektangulärt schema uppdelat i rader och kolumner, där varje cell (angiven av en rad och en kolumn) innehåller ett tal.
- `[X,Y]=meshgrid(x,y)` konstruerar matriser X och Y som innehåller x-koordinaterna respektive y-koordinaterna på punkterna i ett rutnät definierat av vektorerna `x` och `y`.
- Om $f(x, y) = xy$ så definierar $Z = Z.*Y$ en matris som innehåller f:s funktionsvärden på punkterna i rutnätet.
- `mesh(X,Y,Z)` ritar grafen till funktionen vars värden finns i matrisen `Z`.
- `contour(X,Y,Z)` ritar några nivåkurvor till samma funktion.
- `xlabel` och `ylabel` sätter etiketter på koordinataxlarna.
- `title` ger en titel till en figur.

4.5 Övningar

4.1 Rita grafen till $f(x, y) = x^2 - y$ på området $-3 \leq x \leq 3$, och $-5 \leq y \leq 5$. Försök sedan avgöra med papper och penna hur f:s nivåkurvor ser ut. Låt Matlab rita

nivåkurvorna och jämför.

4.2 $g(x) = x^2$ är ju en funktion av en variabel, men samma uttryck ger en funktion av två variabler $g(x, y) = x^2$. Rita grafen till $g(x, y)$, du väljer området själv. Försök att rotera grafen tills du precis ser grafen för funktionen $g(x)$. Utifrån vad du sett så ska du kunna dra slutsatser om hur grafen till $h(x, y) = x^3$ ser ut. skissa först med papper och penna och låt sedan Matlab rita grafen, jämför.

5 Symboliska variabler och symboliska beräkningar

I alla beräkningar och visualiseringar hittills har Matlab använt och producerat numeriska värden, oftast i form av tal på decimalform. Mer exakt har vi hela tiden använt oss av s.k *flyttal* (se dokumentationen för Floating-Point Numbers om du vill läsa mer).

När vi räknar med papper och penna är det ofta betydligt lämpligare att representera tal *exakt* som rationella tal snarare än att skriva ut talens decimalform. Det är också ofta som vi vill göra rent symboliska manipulationer med variabler utan att ge dessa specifika värden; t.ex.säger vi att pq-formeln

$$x = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

ger rötterna till $x^2 + px + q = 0$, utan referens till några konkreta värden på p och q.

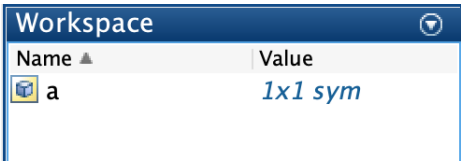
I Matlab kan dessa båda behov tillgodoses med ett och samma hjälpmedel, *symboliska variabler*.

5.1 Matlab som miniräknare återigen

I detta och nästa delavsnitt återgår vi tillfälligt till att arbeta i kommandotolken. Kommandot `sym` kan t.ex. användas som

```
>> a=sym(2/3)

a =

2/3
```

Vi har definierat en symbolisk variabel a med det *exakta* värdet 2/3. Talet representeras som ett rationellt tal och inte, som tidigare, som 0.6666.... Workspacefönstret visar



där `sym` indikerar att a är en symbolisk variabel. Matlab själv refererar till talet `sym(2/3)` som ett *symboliskt tal*, medan jag föredrar att reservera ordet symbolisk för variabler.

I beräkningar med a blir resultatet i allmänhet på nytt exakt, och `ans` blir en symbolisk variabel.

```
>> a^3

ans =

8/27

>> exp(a)

ans =

exp(2/3)

>> a/3

ans =

2/9

>> a*1.275

ans =

17/20
```

Vi ser att om vi blandar a med tal, heltal eller decimaltal i en operation så tar resultatet ett nytt exakt värde genom att decimaltal omvandlas till rationella tal. En funktion som uträknas i a behandlas också som ett exakt värde; observera att `exp(2/3)` inte ersätts med en decimalapproximation. I vissa fall kan Matlab förenkla sådana uttryck, t.ex. om funktionsvärdet har ett känt exakt uttryck.

```
>> p=sym(pi)

p =

pi

>> sin(p)

ans =

0

>> cos(p/6)

ans =
```

```
3^(1/2)/2
```

Vi nöjer oss med detta som illustration av hur Matlab kan fås att använda exakta tal istället för decimaltal med ändligt antal decimaler. Det är lätt att se nyttan med att kunna göra exakta beräkningar, nackdelen är att sådana beräkningar i allmänhet tar mycket längre tid och kräver mer datorkraft.

Om vi i något steg vill uttrycka ett rationellt tal på decimalform använder vi kommandot `eval`.

```
>> b=eval(a)

b =

    0.6667
```

5.2 Symboliska uttryck och funktioner

Att fritt kunna manipulera variabler x, y, etc m h a räknereglerna för tal är något väldigt grundläggande för hur vi gör och diskuterar matematik. I Matlab kan vi göra detta genom att definiera x, y etc som symboliska variabler, antingen en och en via kommandot `x= sym('x')`, ..., eller med kommandot `syms x y z`. En titt på Workspacefönstret visar att vi har definierat tre symboliska variabler med namnen x, y respektive z.

Vi kan skriva uttryck i symboliska variabler med hjälp av de vanliga räkneoperationerna och ge dem som argument till fördefinierade funktioner, vilket i båda fallen resulterar i nya symboliska variabler.

```
>> x^3/(y*z^2)

ans =

x^3/(y*z^2)

>> whos ans
  Name      Size            Bytes  Class    Attributes

  ans       1x1                 8  sym

>> sin(x*y)

ans =

sin(x*y)

>> whos ans
  Name      Size            Bytes  Class    Attributes
```

```
  ans       1x1                 8  sym
```

Observera att produkten mellan x och y måste anges med `x*y`, endast `xy` ger ett felmeddelande (om du inte redan definierat en variabel med namnet xy).

Matlab känner till räkneregler för tal och viktiga egenskaper hos funktioner, men oftast krävs att du instruerar Matlab att använda dessa. Kommandot `simplify` använder kända räkneregler och egenskaper hos funktioner för att försöka förenkla symboliska uttryck, medan kommandot `expand` multiplicerar ihop faktoriserade uttryck och skriver ut termvis.

```
>> (x+2)*(x-2)

ans =

(x-2)*(x+2)

>> simplify(ans)

x^2-4

>> expand((x+y)^2)

ans =

x^2 + 2*x*y + y^2

>> simplify((x+y)^2)

ans =

(x + y)^2

>> sin(x+pi)

ans =

-sin(x)

>> expand(sin(x+y))

ans =

cos(x)*sin(y) + cos(y)*sin(x)

>> simplify( sin(x)^2+cos(x)^2)

ans =

1
```

Som synes känner Matlab till konjugatregeln, kvadreringsregler, additionsformeln för

sinus och trigonometriska ettan. Konjugatregeln tillämpas både med `simplify` och `expand`, eftersom det både följer från att utveckla en produkt samt från att Matlab anser $x^2 - 4$ vara en förenkling av $(x-2)(x+2)$. Kvadreringsregeln leder dock inte till någon förenkling, så `simplify((x+y)^2)` ger ingen förändring. Försök utifrån exemplen ovan att lista ut hur du får Matlab att skriva ut formler för sinus och cosinus av dubbla vinkeln.

Steget är kort från att skriva ett uttryck till att definiera en funktion, så även i Matlab. En *symbolisk funktion* definieras genom att ge ett uttryck i en symbolisk variabel. T.ex. definierar vi den symboliska funktionen $f(x) = x^2$ via `f(x)= x^2`. Funktionen anges nu med `f`, medan funktionsvärdet i en symbolisk variabel y ges av $f(y)$. Speciellt kan f beräknas i exakta tal.

```
>> f(0)

ans =

0

>> f(-2.371)

ans =

5621641/1000000

>> eval(ans)

ans =

    5.6216

>> f(z)

ans =

z^2
```

Symboliska funktioner av flera variabler är lika lätt att definiera, t.ex. definierar `h(x,y)= x^2+y^2` funktionen $h(x, y) = x^2 + y^2$.

Observera att värdet av en symbolisk funktion i ett tal fortfarande blir exakt (ett rationellt tal eller symbolisk variabel). Definiera, som en symbolisk funktion i Matlab, $g(x) = 2^x \sin(x) + 2^{-x} \cos(x)$, och beräkna $g(3.4)$ på decimalform. Om du gjort rätt bör svaret, med fyra decimalers noggrannhet, bli -2.7891.

5.3 Plotta grafer symboliskt

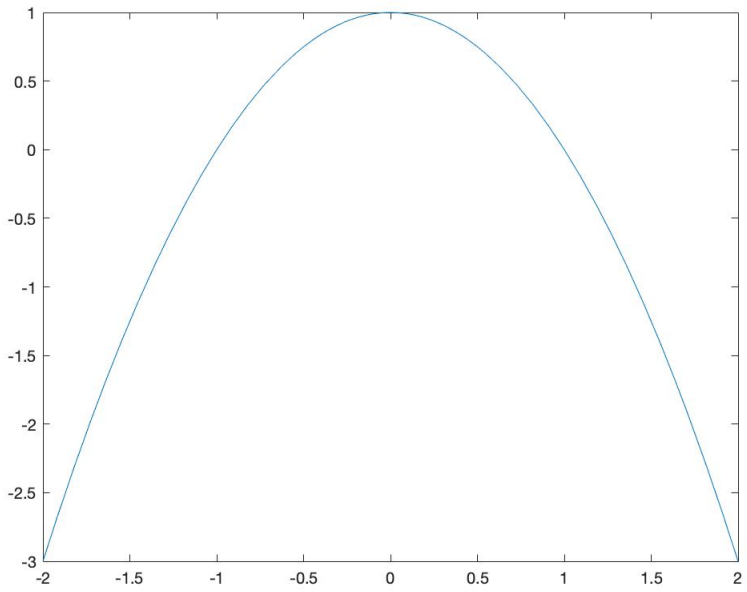
Matlab erbjuder flera sätt att plotta grafen till symboliskt angivna funktioner som komplement till de numeriska plot-funktionerna från avsnitt 3 och 4. Konkret finns komman-

dona `fplot`, `fmesh`, `fsurf`, och `fcontour` som symboliska motsvarigheter till `plot`, `mesh`, `surf`, och `contour`.

Följande script plottar samma graf, visad i Figur 14, som i Figur 6.

```
syms x
f(x)=1-x^2;
fplot(f,[-2,2])
```

Observera att någon punkt innan $\star$ inte behövs eftersom x här inte är en vektor.



Figur 14: Grafen till $f(x) = 1 - x^2$ plottad symboliskt med `fplot` på intervallet $[-2, 2]$.

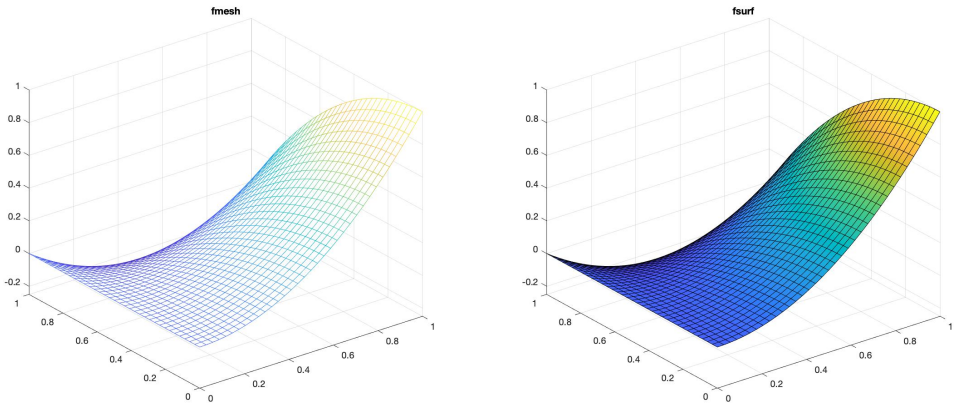
Istället för att först definiera den symboliska funktionen f går det lika bra att endast ange uttrycket i `fplot`, dvs `fplot(x^2,[-1,1])`.

För att plotta $f(x, y) = x^2 - xy^2$ på kvadraten $0 \leq x \leq 1$, $0 \leq y \leq 1$ symboliskt kan vi t.ex. använda följande script.

```
syms x y    %definierar symboliska variabler x och y
f(x,y) = x^2-x*y^2; %definierar den symboliska funktionen (x,y)
%%
subplot(1,2,1)  %delar in figurfonstret i tva delar, "en rad och tva ...
   kolumner", samt later den forsta delen vara aktuell for nasta plot
fmesh(f,[0 1 0 1])  %plottar i det forsta delfonstret m h a fmesh ...
   grafen till f over kvadraten sadan att x och y bada tillhor [0,1]
title('fmesh')  %ger figuren titeln "fmesh"
```

```
%%
subplot(1,2,2)  %later den andra delen vara aktuell for nasta plot
fsurf(f,[0,1,0,1])  %plottar i det andra delfonstret m h a fsurf ...
   grafen till f over kvadraten sadan att x och y bada tillhor [0,1]
title('fsurf')  %ger den andra figuren titeln "fsurf"
```

Den resulterande figuren visas i Figur 15. Jämför Figur 10. I scriptet finns också det nya



Figur 15: Resultatet av att plotta $f(x,y) = x^2 - xy^2$ m h a `fmesh` respektive `fsurf`.

kommandot `subplot`. Kanske är det lätt att gissa hur det fungerar, försök i så fall att testa din gissning genom att rita några grafer. Annars läs kommandots dokumentation.

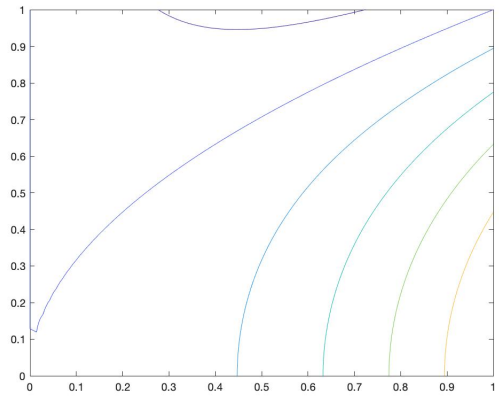
Slutligen använder vi `fcontour` för att symboliskt plotta nivåkurvor.

```
syms x y
f(x,y) = x^2-x*y^2;
fcontour(f,[0,1,0,1])
```

5.4 Inversa funktioner och ekvationer

Matlab kan användas för att lösa ekvationer, vi nöjer oss här med några enkla sådana. Studera först ekvationen $x^2 = 2$, vilken förstås har lösningarna $\pm\sqrt{2}$. Matlab löser denna typ av ekvationer m h a kommandot `solve`. Om x är en symbolisk variabel får vi lösningarna via

```
>> solve(x^2==2,x)

ans =

  2^(1/2)
 -2^(1/2)
```



Figur 16: NIvåkurvor till $(x, y) = x^2 - xy^2$ m h a `fcontour`.

Vi lägger märke till två detaljer; ekvation anges med dubbelt likhetstecken `x^2==2`, och Matlab finner båda rötterna $\pm\sqrt{2}$.

Låt nu p vara polynomet $p(x) = 2x^3 - 3x^2 + 10x - 15$. Vi vill bestämma p:s reella nollställen, och gör detta genom att lösa ekvationen $p(x) = 0$.

```
>> p(x) = 2*x^3-3*x^2+10*x-15;
>> solve(p,x)

ans =

          3/2
 -5^(1/2)*1i
  5^(1/2)*1i
```

Notera att om ekvationen vi vill lösa har högerledet 0 räcker det att ange vänsterledet i `solve`. Vi ser att p har ett reellt nollställe, $x = 3/2$. Vi kan också instruera Matlab att endast finna reella nollställen.

```
>> assume(x,'real')
>> solve(p,x)

ans =

    3/2
```

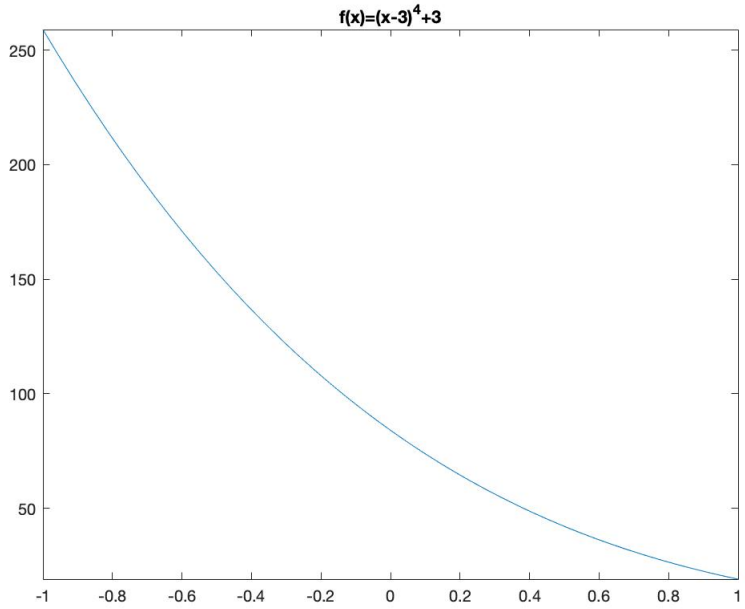
Kommandot `assume` används för att ställa villkor på variabler, och i detta fall är villkoret $x \in \mathbb{R}$. Läs dokumentationen för att se andra exempel på villkor.

Om $f : A \to B$ är en injektiv funktion med värdemängd R så finns en funktion $f^{-1} : R \to A$ sådan att $f(f^{-1}(x)) = x$ och $f^{-1}(f(x)) = x$. Funktionen f^{-1} kallas den inversa funktionen till f, och kan bestämmas genom att lösa ekvationen $y = f(x)$ för

varje $y \in R$, lösningen ges nämligen av $x = f^{-1}(y)$. I Matlab kan vi t.ex. lösa symboliska ekvationer för att bestämma inversa funktioner.

Låt $f(x) = (x-3)^4 + 3$ vara definierad på intervallet $[-1, 1]$. Vi undersöker om f är injektiv, och bestämmer i så fall dess invers. Huruvida f är injektiv kan undersökas genom att studera funktionens graf.

```
syms x
f(x) = (x-3)^4+4;    %definierar funktionen f
fplot(f,[-1,1]) %ritar f:s graf
title('f(x) = (x-3)^4+3')
```



Figur 17: Grafen till $f(x) = (x-3)^4 + 3$ på intervallet $-1 \leq x \leq 1$.

Från Figur 17 är det uppenbart att varje horisontelll linje skär f:s graf i högst en punkt, dvs f är injektiv. Vi försöker därför bestämma f:s invers, vilket innebär att vi vill lösa ekvationen $y = f(x)$ för varje y i f:s värdemängd. Följande script bygger på att variabeln x och funktionen f fortarande finns som variabler i Matlabs minne.

```
syms y
sol=solve(y==f(x),x);
```

Kommandot `solve(y==f(x),x)` säger till Matlab att lösa ut x ur ekvationen $y = f(x)$. Lösningen är värdet på variabeln `sol`:

```
>> sol
```

```
sol =

    (y - 3)^(1/4) + 3
    3 - (y - 3)^(1/4)
 3 - (y - 3)^(1/4)*1i
 3 + (y - 3)^(1/4)*1i
```

Variabeln `sol` är en vektor med fyra komponenter, hur tolkas detta? Ekvationen är en fjärdegradsekvation, och därför förväntar vi oss i allmänhet fyra lösningar. Var och ett av elementen i `sol` är en lösning, dock kan endast en av dessa vara f:s invers. För det första måste f^{-1} endast anta reella värden, så de två sista lösningarna `sol(3)` och `sol(4)` är inte aktuella. För det andra ska värdemängden till f^{-1} vara intervallet $[-1, 1]$ (och definitionsmängden kan vi läsa av ungeärligt från Figur 17). Vi beräknar värdet på uttrycken `sol(1)` och `sol(2)` i $y = 250$, som uppenbarligen tillhör definitionsmängden till den inversa funktionen.

```
>> eval(subs(sol(1),y,250))

ans =

    6.9644

>> eval(subs(sol(2),y,250))

ans =

    -0.9644
```

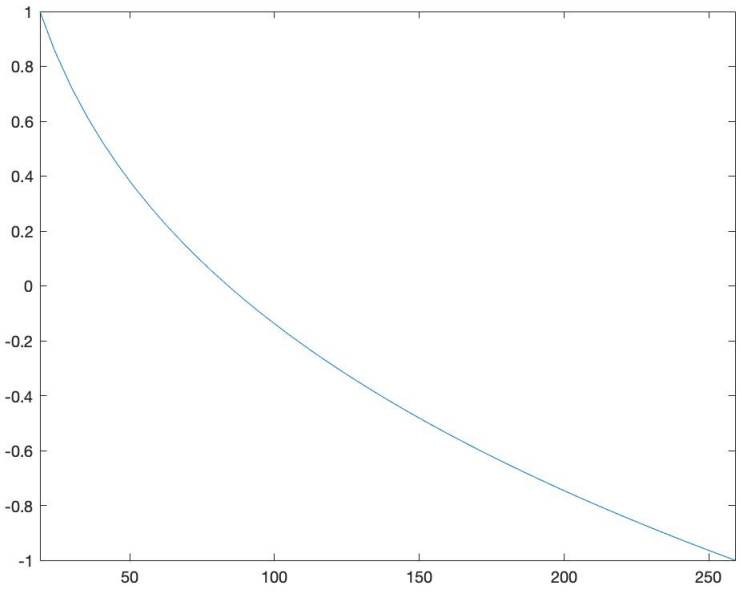
Endast `sol(2)`:s värde tillhör f^{-1}:s värdemängd, alltså är detta f:s invers. Kommandot `subs` används för att ersätta en symbolisk variabel med en annan, i det här fallet ersätts y med det symboliska (dvs exakta) talet 250. Vi definierar f^{-1} och plottar dess graf med följande script.

```
finv(x) = subs(sol(2),y,x); %definierar f:s invers som funktion av x
lowlim=eval(f(1));  %nedre gransen av definitionsmangden till f:s invers
uplim=eval(f(-1));  %ovre gransen av definitionsmangden till f:s invers
fplot(finv,[lowlim,uplim])
```

I stället för att först bestämma alla lösningar till en ekvation för att sedan försöka plocka ut vilken som svarar mot just f^{-1} kan vi sätta villkor på x och y för att direkt få rätt lösning. Vi använder att f:s definitionsmängd är $[-1, 1]$ medan dess värdemängd är `[lowlim,uplim]` $= [19, 259]$.

```
assume(-1≤x & x≤1)
assume(19≤y & y≤259)
sol=solve(y==f(x),x);
```

Observera att tecknet $\leq$ skrivs `<=`. Kontroll ger



Figur 18: Grafen till f^{-1}.

```
>> sol

sol =

3-(y-3)^(1/4)
```

vilket stämmer överens med vad den förra metoden gav.

5.5 Sammanfattning

- `sym` och `syms` definierar symboliska variabler, t.ex. `a=sym(1/5)` eller `syms x`. Talet a är vad Matlab kallar ett *symboliskt tal*, och används som ett exakt uttryck (i detta fall rationellt tal) snarare än decimaltal.
- `eval` ersätter ett symboliskt tal med sin decimalform (eller mer exakt, sin flyttalsapproximation).
- `simpliy` och `expand` använder räkneregler och funktionsegenskaper för att förenkla respektive utveckla symboliska uttryck.
- `fplot`, `fmesh`, `fsurf`, och `fcontour` är symboliska motsvarigheter till `plot`, `mesh`, `surf`, och `contour` respektive.
- `subplot` delar in figurfönstret i flera delar.
- Ekvationer löses med kommandot `solve`, t.ex. `solve(x^2==2,x)`.
- Villkor på symboliska variabler anges med `assume`.

5.6 Övningar

Du ska lösa minst tre av uppgifterna nedan, men valet av uppgifter måste uppfylla två villkor.

1. Du måste lösa minst en av 5.1 och 5.2, minst en av 5.3 och 5.4, samt minst en av 5.5 och 5.6.

2. Om du inte löser 5.4 så ska du lösa 5.6

5.1 Bestäm exakt värdena $\sin(\alpha)$ och $\cos(\alpha)$ för $\alpha = \pi/5$, $\pi/8$, och $\pi/12$.

5.2 Med hjälp av de Moivres formel är det inte svårt att uttrycka $\sin(n\theta)$ och $\cos(n\theta)$ i termer av $\sin(\theta)$ respektive $\cos(\theta)$, och det är förstås ännu enklare att låta Matlab göra detta. Bestäm sådana uttryck för $n = 5, 6, 7, 8$.

5.3 Låt $h(x, y) = (x^2 + y^2)(x^2 + y^2 - 1)$. Rita grafen och några nivåkurvor till h i kvadraten $-0.7 \leq x \leq 0.7$, $-0.7 \leq y \leq 0.7$. I figuren ska finnas två delfigurer, den ena med funktionens graf och den andra med nivåkurvorna.

5.4 För varje tal $a > 1$, låt sh_a, ch_a, respektive $\mathrm{th}_a(x)$ vara funktionerna

$$\mathrm{sh}_a(x) = \frac{1}{2}\left(a^x - a^{-x}\right), \ \mathrm{ch}_a(x) = \frac{1}{2}\left(a^x + a^{-x}\right), \ \mathrm{th}_a(x) = \frac{\mathrm{sh}_a(x)}{\mathrm{ch}_a(x)}.$$

För $a = e$ blir dessa funktioner precis de *hyperboliska funktionerna* $\sinh(x)$, $\cosh(x)$, och $\tanh(x)$, se t.ex. [Calculus1, sid. 107–112]. För $a = 2$, $a = e$ och $a = 10$, rita graferna till funktionerna. Graferna till de tre sh_a-funktionerna ska ritas i samma figur, markerade så att de kan identifieras, och detsamma gäller ch_a respektive th_a. Totalt ska du alltså producera tre figurer med tre grafer i vardera figur.

5.5 Lös m h a Matlab uppgift 224 i [Calculus1, Kap. 1].

5.6 Välj en av funktionerna i uppgift **5.4** för något tal $a > 1$, och begränsa om nödvändigt definitionsmängden så att du får en injektiv funktion. Bestäm inversen till denna funktion. Rita graferna till funktionen och dess invers i samma figur.

6 Matriser, vektorer, och linjära ekvationssystem

Namnet *Matlab* kommer från Matrix laboratory, så det bör inte vara någon överraskning att det är just matrisberäkningar som är Matlabs styrka. Det är oftast möjligt att utföra en uppgift i Matlab på många olika sätt, och om det är möjligt att utföra den med hjälp av matriser och de vanliga matrisoperationerna så är detta oftast det mest effektiva sättet.

6.1 Definiera och räkna med matriser och vektorer

Du har redan sett några olika metoder att konstruera vektorer i Matlab via `t=0:0.1:1`, `b=linspace(0,1,50)` och `a=[0,1,-3,2]`. I det sista fallet går det lika bra att utelämna kommatecknen: `a=[0 1 -3 2]`, vilket återspeglar hur Matlab själv tolkar uttrycket: som en radvektor, dvs en $1 \times k$-matris (här är förstås $k = 4$). Kolonnvektorer kan definieras direkt genom att avgränsa elementen med semikolon `aT=[0; 1; -3; 2]`, eller genom att transponera radvektorer:

```
>> transpose([0 1 -3 2])

ans =
     0
     1
    -3
     2
```

Det finns även ett kortkommando för `transpose`, nämligen `'` eller egentligen `.'`. För matriser vars element är reella tal finns ingen skillnad mellan de två kortkommandona, men i allmänhet är resultatet olika.

Matriser kan definieras konkret på liknande sätt, där rader åtskiljs med semikolon. T.ex. definierar

```
>> M=[1 2 3 4;5 6 7 8;4 3 2 1]

ans =

M =

     1     2     3     4
     5     6     7     8
     4     3     2     1
```

förstås 3×4-matrisen

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 4 & 3 & 2 & 1 \end{pmatrix}.$$

Vi ser från

```
>> whos
  Name        Size              Bytes  Class     Attributes

  M           3x4                  96  double
  a           1x4                  32  double
  aT          4x1                  32  double
  b           1x50                400  double
  t           1x11                 88  double
```

att M är en 3×4-matris, att a är en 1×4-matris (radvektor med 4 element), aT är en 4×1-matris (kolonnvektor med 4 element), osv.

Enskilda element i matriser anges med rad och kolonnnummer

```
>> M(2,3)

ans =

    7
```

men det går också att ange flera element samtidigt. Följande exempel är förhoppningsvis självförklarande.

```
>> M([1 3 2],2)

ans =

     2
     3
     6

>> M(2,2:4)

ans =

     6     7     8

>> m=M(1:3,2:4)

m =

     2     3     4
     6     7     8
     3     2     1

>> whos  m
  Name      Size             Bytes  Class     Attributes

  m         3x3                 72  double

>> size(M)

ans =

     3     4

>> M(3,:)

ans =
```

```
     4     3     2     1
```

Vi kan utöka matriser genom att lägga till element

```
>> M(2,6)=3

M =

     1     2     3     4     0     0
     5     6     7     8     0     3
     4     3     2     1     0     0
```

(notera att Matlab automatiskt lägger till 0 i alla element som inte är direkt specificerade för att få en matris) eller genom att stapla matriser horisontellt eller vertikalt

```
>> [M M]

ans =

     1     2     3     4     0     0     1     2     3     4     0     0
     5     6     7     8     0     3     5     6     7     8     0     3
     4     3     2     1     0     0     4     3     2     1     0     0

>> [M;M]

ans =

     1     2     3     4     0     0
     5     6     7     8     0     3
     4     3     2     1     0     0
     1     2     3     4     0     0
     5     6     7     8     0     3
     4     3     2     1     0     0
```

Det finns funktioner för att skapa speciella typer av matriser; `zeros` och `ones` t.ex. definierar nollmatriser respektive matriser bara bestående av ettor, medan `rand` definierar matriser med slumptal mellan 0 och 1.

```
>> ones(5,3)

ans =

     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

Se dokumentationen för mer information om hur dessa funktioner används. Identitetsmatrisen skapas via `eye`:

```
>> eye(5)

ans =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

De vanliga matrisoperationerna, addition (subtraktion), skalärmultiplikation, och multiplikation, anges med symbolerna $+(-)$ och $*$.

```
>> m+m

ans =

     4     6     8
    12    14    16
     6     4     2

>> -3*m

ans =

    -6    -9   -12
   -18   -21   -24
    -9    -6    -3

>> m*M

ans =

    33    34    35    36     0     9
    73    78    83    88     0    21
    17    21    25    29     0     6
```

Som du redan har sett i fallet radvektorer så beräknar Matlab funktionsvärden elementvis om du ger en matris som argument till en fördefinierad funktion.

```
>> sin(M)

ans =

    0.8415    0.9093    0.1411   -0.7568         0         0
   -0.9589   -0.2794    0.6570    0.9894         0    0.1411
   -0.7568    0.1411    0.9093    0.8415         0         0
```

Elementvisa aritmetiska operationer anges, precis som tidigare, med en punkt före operationen

```
>> M.*M

ans =

     1     4     9    16     0     0
    25    36    49    64     0     9
    16     9     4     1     0     0

>> m.^3

ans =

     8    27    64
   216   343   512
    27     8     1

>> m./m^2

ans =

    0.0588    0.0857    0.1111
    0.0769    0.0843    0.0909
    0.1429    0.0800    0.0345
```

6.2 Linjära ekvationssystem och matrisekvationer

Betrakta ett allmänt ekvationssystem med r ekvationer och k obekanta.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1k}x_k &= b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2k}x_k &= b_2 \\ & \vdots \\ a_{r1}x_1 + a_{r2}x_2 + ... + a_{rk}x_k &= b_r \end{cases} \tag{6.1}$$

Till ekvationssystemet hör tre matriser, koeffficientmatrisen A, totalmatrisen T, och "högerledet" $\vec{b}$.

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1k} \\ a_{21} & a_{22} & ... & a_{2k} \\ \vdots & & \vdots & \vdots \\ a_{r1} & a_{r2} & ... & a_{rk} \end{pmatrix}, \quad T = \left(\begin{array}{cccc|c} a_{11} & a_{12} & ... & a_{1k} & b_1 \\ a_{21} & a_{22} & ... & a_{2k} & b_2 \\ \vdots & & \vdots & \vdots & \vdots \\ a_{r1} & a_{r2} & ... & a_{rk} & b_r \end{array}\right), \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix}$$

Gauss-Jordanmetoden för att lösa (6.1) går ut på att via elementära radoperationer föra totalmatrisen T till en radkanonisk matris R. Matlab gör detta med kommandot `rref` (förkortning för *Reduced Row Echelon Form*, vilket är engelska för radkanonisk form).

Ekvationssystemet

$$\begin{cases} -2x_1 + x_2 - x_3 + 3x_4 &= 1 \\ 3x_1 + 3x_2 + 3x_3 &= 3 \\ -x_1 - 4x_2 + x_3 + x_4 &= 5 \\ -2x_1 - x_2 \qquad - 2x_4 &= 2 \end{cases} \tag{6.2}$$

löses med följande script.

```
T=[-2 1 -1 3 1;3 3 3 0 3;-1 -2 1 1 5;-2 -1 0 -2 2]; %systemets totalmatris
R=rref(T)   %radreducera T till radkanonisk form R
```

Scriptet ger radkanonisk form

```
R =

    1.0000         0         0         0   -1.2903
         0    1.0000         0         0   -0.3226
         0         0    1.0000         0    2.6129
         0         0         0    1.0000    0.4516
```

vilket innebär att systemet har den entydiga lösningen

$$(x_1, x_2, x_3, x_4) = (-1.2903, -0.3226, 2.6129, 0.4516). \tag{6.3}$$

Vi vet att ekvationssystemet (6.1) är ekvivalent med matrisekvationen $A\vec{x} = \vec{b}$, där $\vec{x}$ är kolonnvektorn med de obekanta x_1, x_2, x_3, x_4. Matlab löser sådana matrisekvationer med `A\b`, där \ är tänkt att påminna om division "från vänster", dvs $A^{-1}\vec{b}$. Följande script löser (6.2)

```
%   systemets koefficientmatris A:
A=T(:,1:4);
%   hogerledet b:
b=T(:,5);
%   losning till Ax=b:
x=A\b
```

med resultat

```
x =

   -1.2903
   -0.3226
    2.6129
    0.4516
```

Lösningen är, åtminstone med fyra decimalers noggrannhet, densamma som Gauss-Jordanmetoden gav. Båda metoder har fördelar och nackdelar. Radreduktion, `rref`, är mindre effektiv än \ vilket kan spela stor roll vid väldigt stora ekvationssystem. Den

senare metoden fungerar visserligen bra för inverterbara matriser A, men ger inte samma information vid icke-inverterbara koefficientmatriser. Om A är kvadratisk men inte inverterbar ger metoden otillförlitliga svar, och ibland icke-numeriska svar som härrör från division med noll. Om A inte är kvadratisk så producerar metoden en s.k. "minsta kvadrat-lösning" till ekvationssystemet. Om ekvationssystemet har lösningar så är minsta kvadratlösningen en av dem, och om ekvationssystemet är inkonsistent är det en slags bästa approximation till en lösning. Läs gärna mer om minsta kvadratlösningar i dokumentationen för `lsqminnorm`.

Matrisekvationen $A\vec{x} = \vec{b}$ kan förstås också lösas genom att invertera A och multiplicera med $\vec{b}$. Den inbyggda funktionen för matrisinvers heter `inv`, men det fungerar också att använda Jacobis metod. Enligt Matlabs dokumentation är `A\b` mer robust än `inv(A)*b`, vilket innebär att den typiskt producerar mindre numeriska fel. Vi testar ändå.

```
>> inv(A)*b

ans =

   -1.2903
   -0.3226
    2.6129
    0.4516
```

Igen samma svar med fyra decimalers noggrannhet.

Alla metoder ovan ger svar på decimalform, medan en lösning för hand ju producerar en exakt lösning om elementen i A och $\vec{b}$ är exakt givna. En rationell lösning som approximeras av vektorn x kan bestämmas via kommandot `sym`.

```
>> sym(x)

ans =

 -40/31
 -10/31
  81/31
  14/31
```

Den resulterande vektorn är den exakta lösningen till ekvationssystemet vilket verifieras genom att multiplicera med koefficientmatrisen.

```
>> A*sym(x)

ans =

 1
 3
 5
```

```
 2
```

Det finns dock ingen garanti att denna metod ger korrekt svar, för att säkerställa detta kan systemet istället lösas med exakta koefficienter och högerled.

```
exactT=sym(T);
exactR=rref(exactT)
```

ger resultatet

```
exactR =

[ 1, 0, 0, 0, -40/31]
[ 0, 1, 0, 0, -10/31]
[ 0, 0, 1, 0,  81/31]
[ 0, 0, 0, 1,  14/31]
```

Att alltid jobba med exakta, alltså symboliska, matriser garanterar att erhållna lösningar är korrekta och exakta. Dock kan det kräva betydligt mer tid och datorkraft att låta Matlab arbeta med symboliska matriser än med flyttalsmatriser.

Eftersom Matlab kan lösa linjära ekvationssystem med symboliska totalmatriser så är det förstås möjligt att också lösa ekvationssystem som har obestämda parametrar. Betrakta t.ex.

$$\begin{cases} kx + 2y &= 1 \\ x + 3y &= 2 \end{cases}$$

där k är ett reellt tal. Vi löser m h a \.

```
syms k
A = [k 2;1 3];
b=sym([1;2]);
x=A\b
```

Scriptet ger

```
x =

        -1/(3*k - 2)
 (2*k - 1)/(3*k - 2)
```

Radreducering via `rref` ger samma resultat (kontrollera detta). Här ser vi att vi måste vara försiktiga med att tolka resultatet; svaret har uttrycket $3k - 2$ i nämnaren vilket är noll då $k = 2/3$. En kontroll av matrisen A:s determinant ger

```
>> det(A)

ans =
```

```
3*k - 2
```

så det "farliga" värdet på parametern k svarar alltså mot en icke-inverterbar koefficientmatris. Den angivna lösningen är ok för alla andra värden på k, men fallet $k = 2/3$ måste hanteras separat.

```
>> rref([subs(A,k,2/3) b])

ans =

[ 1, 3, 0]
[ 0, 0, 1]
```

Med andra ord, ekvationssystemet saknar lösning för $k = 2/3$.

6.3 Linjära avbildningar, egenvärden och egenvektorer, samt geometriska egenskaper hos vektorer

Eftersom vi kan multiplicera matriser så kan vi också arbeta med linjära avbildningar. Kom ihåg att om A är en $m \times n$-matris så är den linjära avbildningen T_A med standardmatris A en funktion med definitionsmängd $\mathbb{R}^n$ (vilket vi identifierar med mängden av alla kolonnvektorer med n element) och målmängd $\mathbb{R}^m$ (mängden av alla kolonnvektorer med m element). Mer precist är T_A den funktion som i kolonnvektorn $\vec{x} = (x_1\ x_2\ ...\ x_n)^T$ tar värdet $A\vec{x}$.

Låt

$$A = \begin{pmatrix} 1 & 3 \\ 3 & -1 \end{pmatrix}.$$

och betrakta den linjära avbildningen T_A. Vi prövar att beräkna värdet av T_A i ett antal olika kolonnvektorer, säg

$$\vec{u}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ \vec{u}_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix},\ \vec{e}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ \vec{e}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

```
A=[1 3;3 -1];
u1=[1;1]; u2=[1;2];
e1=[1;0]; e2=[0;1];
v1=A*u1
v2=A*u2
w1=A*e1
w2=A*e2
```

Scriptet ovan resulterar i

```
v1 =

     4
```

```
     2


v2 =

     7
     1


w1 =

     1
     3


w2 =

     3
    -1
```

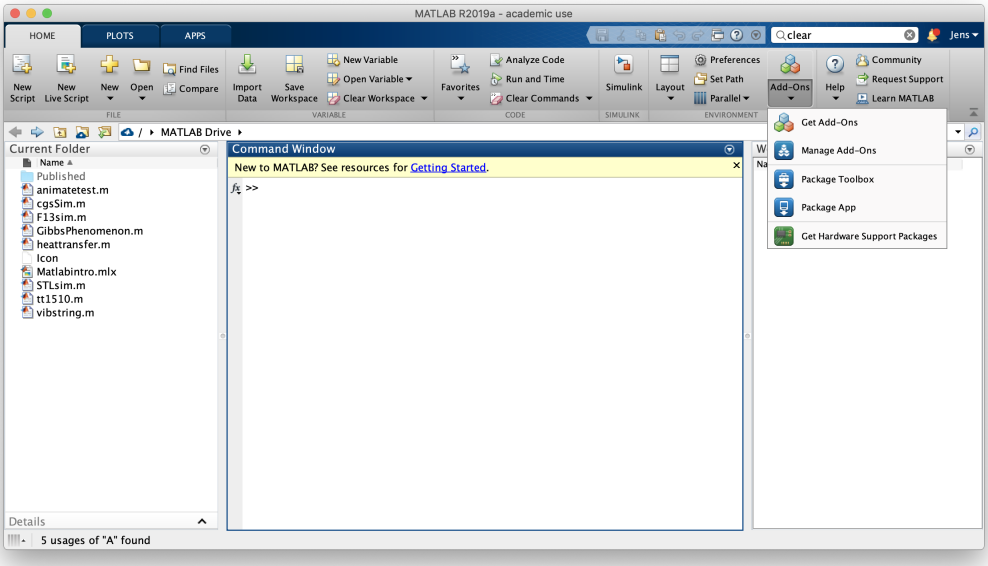
Det är ofta användbart att illustrera vektorer med två och tre element, och vi kan göra detta på flera olika sätt i Matlab. Det går t.ex. att använda `plot` och `plot3` i två respektive tre dimensioner, eller funktionerna `quiver` och `quiver3`. Det enklaste sättet att lära sig hur det går till är att googla "draw vectors matlab" eller liknande. På MathWorks hemsida finns en också en flik "Community" där du kan söka svar på liknande frågor som ställts tidigare. Du får själv ta ansvaret för att sätta dig in i något sätt att illustrera vektorer i två och tre dimensioner. I den här texten kommer jag att använda mig av funktionen `drawVector` som inte är inbyggd i Matlab, men som finns att ladda ner till din egen dator via MathWorks hemsida.

Om du själv vill använda samma funktioner så följer du instruktionerna här nedan. Kontrollera först om du redan har funktionen `drawVector` installerad genom att skriva `doc drawVector` i Kommandotolken. Om du får upp en hjälpfil för kommandot så har du redan funktionen och behöver inte göra mer, om inte så behöver du installera funktionen som följer. Under *Home*-fliken på Matlabfönstret finns en ikon med namnet *Add-Ons*. Klickar du på den kan du välja mellan olika alternativ, välj *Get Add-Ons*. Detta öppnar en browser kallad *Add-On Explorer* där du kan söka efter olika typer av funktioner och andra tillägg som Matlabanvändare gjort tillgängliga. Sök efter *drawLA*, och öppna sökresultatet "drawLA - Draw Toolbox for Linear Algebra". Väl där klickar du *Add* så kommer Matlab att installera en ny uppsättning funktioner. Eventuellt behöver du logga in, och har du inte redan registrerat dig som användare behöver du då först göra detta. Nu bör du ha funktionen `drawVector` tillgänglig, vilket du kan testa på samma sätt som ovan.

För att plotta vektorerna $\vec{u}_1$, $\vec{u}_2$, $\vec{e}_1$, och $\vec{e}_2$ kan vi t.ex. ge kommandot

```
>> drawVector([u1 u2 e1 e2],{'u1','u2','e1','e2'})
```

Resultatet visas i Figur 20. Som synes tar `drawVector` en lista (dvs en vektor) av



Figur 19: Kommandot *Get Add-Ons* öppnar ett browserfönster där du kan söka tillägg till Matlab

kolonnvektorer som argument. Den andra listan, avgränsad med krullparenteser, anger etiketter för var och en av vektorerna. För var och en av kolonnvektorerna så ritar kommandot ut en punkt vars koordinater är vektorns element, samt en linje från origo till punkten. Kommandot klarar inte att rita ut pilspetsar (dock en grov approximation med tillägget `'b>-'`), så den vanliga pilrepresentationen av vektorer ser vi inte. Lägg nu till i samma figur värdet av T_A i dessa vektorer, vi gör det i en annan färg.

```
>> hold on
>> drawVector(v1 v2 w1 w2],{'v1','v2','w1','w2'},'ro-')
```

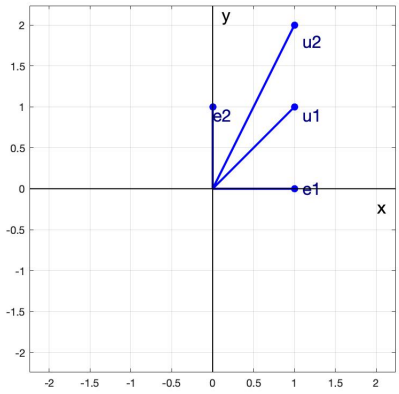
Kommandot `drawVector` fungerar också i tre dimensioner som följande script illustrerar.

```
u=[1;1;1];
B=[sqrt(3)/2 -1/2 0;1/2 sqrt(3)/2 0;0 0 1/2];
drawVector([u B*u B*B*u B*B*B*u B*B*B*B*u])
```

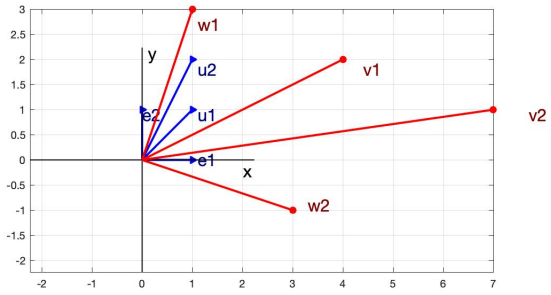
Resultatet visas i Figur 22.

Matlab har ett kommando, `eig`, för att bestämma både egenvärden och egenvektorer till matriser. T.ex. beräknas egenvärdena till matrisen $\left(\begin{smallmatrix}1&2\\1&3\end{smallmatrix}\right)$ som följer.

```
>> eig([1 2;1 3])
```



Figur 20: Resultatet av att låta `drawVector` plotta vektorer.

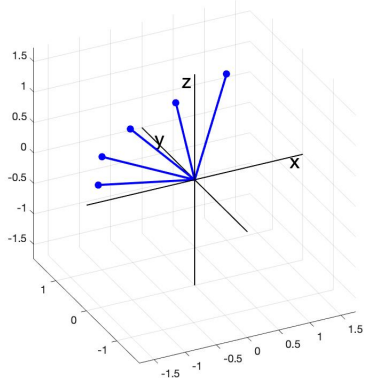


Figur 21: Med $T_A(\vec{u}_1)$ etc. utritade i samma figur.

```
ans =

    0.2679
    3.7321
```

För de exakta värdena anges istället motsvarande symboliska matris

```
>> eig(sym([1 2;1 3]))

ans =

 2 - 3^(1/2)
 3^(1/2) + 2
```



Figur 22: Vektorn $\vec{u} = (1\ 1\ 1)^T$ utritad tillsammans med $B\vec{u}$, $B^2\vec{u}$, $B^3\vec{u}$, och $B^4\vec{u}$.

Samma kommando men använt på ett lite annat sätt ger också matrisens egenvektorer.

```
>> [V,D] = eig(sym([1 2;1 3]))

V =

[ - 3^(1/2) - 1, 3^(1/2) - 1]
[             1,           1]


D =

[ 2 - 3^(1/2),           0]
[           0, 3^(1/2) + 2]
```

Här ger kommandot två st matriser som resultat. Den första matrisen, som vi gett namnet V ovan, har egenvektorer till matrisen som kolonner. Den andra matrisen, D, är en diagonalmatris med egenvärdena i huvuddiagonalen; ordningen på egenvärdena är samma som ordningen hos kolonnerna i matrisen V, dvs den första kolonnen i V är egenvektor med egenvärdet i den första kolonnen i matrisen D, osv. Vi kontrollerar lätt detta med följande script

```
A = sym([1 2;1 3]); %matrisen vars egenv\"arden och egenvektorer vi ...
   vill best\"amma
[V,D] = eig(A); %egevektorer och egenvarden till A
v1=V(:,1); %kolonn nr 1 i V
v2=V(:,2); %kolonn nr 2 i V
a=A*v1; % A multiplicerad med egenvektor 1
b=A*v2; %A multiplicerad med egenvektor 2
simplify(a-D(1,1)*v1) %differensen mellan A v1 och lambda1 v1
simplify(b- D(2,2)*v2) %differensen mellan A v2 och lambda2 v2
```

Resultatet blir

```
ans =

 0
 0

 ans =

 0
 0
```

vilket visar att $A\vec{v}_1 = (D)_{11}\vec{v}_1$ och $A\vec{v}_2 = (D)_{22}\vec{v}_2$, dvs att första och andra kolonnerna i matrisen V är egenvektorer till A med egenvärden givna av diagonalelementen i matrisen D. Allt detta kan givetvis utföras med flyttal istället för exakta värden, tag bara bort kommandona `sym` och `simplify` i scriptet ovan.

Normen av en vektor (som anges m h a koordinater i en ON-bas) beräknas med `norm`, och skalärprodukten med `dot`. Följande exempel får illustrera användningen.

```
>> norm(1;2;1)

ans =

    2.4495


>> dot([1;2;3],[1;-2,1])

ans =

     0
```

Det sista exemplet kan naturligtvis också beräknas som matrisprodukten $\begin{pmatrix}1&2&3\end{pmatrix}\begin{pmatrix}1\\-2\\1\end{pmatrix}$. Testa själv att göra ovanstående beräkningar exakt istället för med flyttal.

I tre dimensioner beräknas vektorprodukten (kryssprodukten) $\vec{u} \times \vec{v}$ av två vektorer angivna i en högerhänt ON-bas med kommandot `cross(u,v)`. Scriptet

```
u=[1;2;3];
v=[1;0;2];
w=cross(u,v)
```

ger resultatet

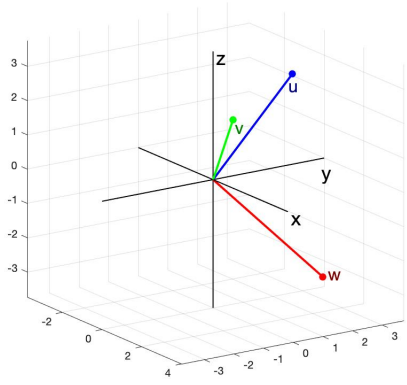
```
w =

     4
```

```
     1
    -2
```

och vi verifierar relationen mellan de tre vektorerna grafiskt

```
>> drawVector(u,{'u'})
>> hold on
>> drawVector(v,'go-',{'v'})
>> drawVector(w,'ro-',{'w'})
```

med resultat visat i figur 23.



Figur 23: Vektorerna $\vec{u}$, $\vec{v}$, och $\vec{w} = \vec{u} \times \vec{v}$ utritade i samma figur.

6.4 Sammanfattning

- Små matriser kan definieras för hand, t.ex. definieras 2×3-matrisen $M = \left(\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{smallmatrix}\right)$ via `M=[1 2 3;4 5 6]`.
- `ones` definierar matriser med bara ettor, `zeros` ger nollmatriser.
- `eye(n)` ger identitetsmatrisen av ordning n.
- `size` ger typen hos en matris
- `transpose(M)`, `M.'`, och `M'` ger transponatet av M (med reella element, annars ger det sista exemplet det s.k. *Hermitekonjugatet* av M).
- `M(i,j)` ger elementet i rad nr i och kolonn nr j i matrisen M, och mängder av matriselement plockas ut genom att ange mängder av rad respektive kolonnindex t.ex. enligt `M(1:4,3)` som ger en kolonnvektor med elementen i raderna 1–4 och kolonn 3 ur matrisen M.
- Matrisaddition, skalärmutliplikation, och matrismultiplikation anges med + respektive `*`.
- Elementvis matrismultiplikation anges med `.*`.

- `rref` utför radreducering till radkanonisk form, och `Ab` löser matrisekvationen $A\vec{x} = \vec{b}$.
- `inv` beräknar matrisinvers, och `det` beräknar determinant.
- `drawVector` (ej inbuggt i Matlab, måste installeras) ritar vektorer i två och tre dimensioner.
- `eig` bestämmer egenvärden och egenvektorer.
- `norm`, `dot`, och `cross` beräknar norm, skalärprodult, respektive vektorprodukt av vektorer i en ON-bas (i det sista fallet högerorienterad bas för vektorerna i rummet).

6.5 Övningar

6.1 Du ska här lösa underbestämda och överbestämda ekvationssystem.

(a) Låt Matlab lösa ekvationssystemet

$$\begin{cases} x + y + z &= 1 \\ 2x + y &= 0 \end{cases}$$

exakt (dvs symboliskt) m h a kommandot \. Lös sedan systemet med papper och penna och skriv ner den allmäna lösningen på parameterform. Finn parametervärdet som motsvarar Matlabs lösning.

(b) Låt Matlab lösa ekvationssystemet

$$\begin{cases} x + y &= a \\ x - y &= 1 \\ 2x + y &= 2 \end{cases}$$

symboliskt m h a \. Här representerar a ett godtyckligt tal. Får du en lösning? I så fall vad betyder den? Om inte, använd Matlab för att undersöka för vilka värden på a som ekvationssystemet är konsistent, och lös systemet för de värdena på a.

6.2 Lös ekvationssystemet

$$\begin{cases} 6x_1 + x_2 + 3x_4 &= -3 \\ -9x_1 + 2x_2 + 3x_3 - 8x_4 &= 1 \\ x_1 - 4x_3 + 5x_4 &= 2 \end{cases}$$

m h a Matlab. Använd först `rref`, och sedan kommandot \. Ger de två metoderna konsekventa resultat? Förklara hur de är relaterade.

6.3 Betrakta rotationsmatrisen

$$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}.$$

Låt Matlab bestämma egenvärdena för $\theta = \pi/6$ respektive för $\theta = \pi$. Förklara i det första fallet resultatet genom att lös den karakteristiska ekvationen antingen för hand eller m h a Matlab (lös den dock exakt, ej numeriskt). Vad ger Matlab för egenvektorer i det fallet? Förklara med ett geometriskt resonemang varför matrisen inte har några egenvektorer i planet. Förklara på liknande sätt resultatet i det andra fallet ($\theta = \pi$).

6.4 Låt

$$T = \frac{1}{7}\begin{pmatrix} 6 & -2 & -3 \\ -2 & 3 & -6 \\ -3 & -6 & -2 \end{pmatrix}$$

Matrisen T beskriver en avbildning av vektorerna i rummet, och din uppgift är att beskriva geometriskt vad matrisen gör med kolonnvektorer. Bestäm först egenvärden och egenvektorer till T. Utifrån resultatet följer att vissa vektorer är lätta att beskriva vad som sker med när du multiplicerar med T. Illustrera vad du har funnit genom att rita några vektorer samt T multiplicerad på samma vektorer. Finns det andra saker du kan illustrera i Matlab för att visa tydligt hur T verkar geometriskt? Kan du t.ex. använda kommandot `drawSpan`?

Kan du finna en bas till vektorerna i rummet där basvektorerna påverkas av T på ett lättbegripligt sätt? Förklara utifrån en sådan bas hur T verkar på en godtycklig vektor $\vec{v}$ i rummet.

6.5 Bestäm numeriskt i grader vinkeln mellan vektorerna

$$\vec{u} = \begin{pmatrix} -1 \\ 2 \\ 5 \end{pmatrix}, \quad \vec{v} = \begin{pmatrix} 4 \\ -3 \\ 3 \end{pmatrix}$$

och bestäm sedan $\vec{u}$:s komposanter vinkelrätt mot och parallellt med $\vec{v}$. Rita upp $\vec{u}$, $\vec{v}$, och komposanterna i en och samma figur. Verifiera rimligheten i ditt resultat för vinkeln genom att uppskatta dess värde (rotera figuren till du kan se vinkeln tydligt).

6.6 Du ska använda Matlab för att beräkna avståndet från ett plan till en punkt S, samt den punkt på planet som är närmast punkten S. Planet är det plan som innehåller punkterna $P : (10, -2, -2)$, $Q : (2, 5, -1)$, och $R : (-2, -1, 3)$, och punkten vars avstånd till planet ska bestämmas är $S : (5, 5, -4)$.

Referenser

[Calculus1] E. Herman, G. Strang, et al., *Calculus Volume 1*, openstax (2016)
https://openstax.org/details/books/calculus-volume-1